Broadview
www.broadview.com.cn

以任务驱动方式讲解，用实例引导读者学习
只需21天，便可轻松掌握Linux C编程

畅销丛书
16万册

21天学通 Linux C编程

(第2版)

● 弓雷　编著 ●



本书特色

- 基础知识→核心技术→典型实例→综合练习→项目案例
- 218个典型实例、1个案例、59个练习题、55个面试题
- 一线开发人员全程贴心讲解，上手毫不费力

超值DVD

- 21小时多媒体语音视频教学
- 本书源代码 + 本书电子教案（PPT）

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

21天学编程系列

内 容 简 介

本书由浅入深、通俗易懂地讲解了 Linux 系统下的 C 程序设计。全书共 21 章，从 Linux 系统安装开始，讲解了 Linux 常用命令、shell 编程、C 语言基础、指针与函数、C 程序网络编程、图形界面程序设计等内容。书中通过大量的实例来讲解知识要点，读者可以通过这些实例和练习系统地学习 Linux 下的 C 程序设计，从而提高自己的编程能力。

本书共分七篇。第一篇为 Linux 基础篇，介绍 Linux 编程平台的搭建及文本编辑器的使用；第二篇为 shell 编程篇，介绍 Linux 下的常用 shell 命令与 shell 编程；第三篇为 C 程序基础篇，介绍 C 程序设计的基础知识；第四篇为指针与函数篇，重点介绍 C 语言中指针与函数的概念和用法；第五篇为文件与目录篇，介绍 C 语言中的文件与目录操作功能；第六篇为网络编程篇，介绍网络编程的基础知识和如何使用 C 语言进行网络编程；第七篇为图形界面开发篇，介绍基于 Linux 的 C 程序图形界面开发及 Glade 的使用。

本书语言通俗易懂，内容丰富，注重理解与实例，知识涵盖面广，非常适合 C 语言的初学者、高校相关专业学生、程序开发人员、Linux 工作人员阅读和学习。

本书附赠 DVD 光盘 1 张，内容包括超大容量手把手教学视频、电子教案（PPT）、源代码及各章习题答案。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

21 天学通 Linux C 编程 / 弓雷编著. —2 版.—北京：电子工业出版社， 2012.7
（21 天学编程系列）
ISBN 978-7-121-17151-2

Ⅰ. ①2… Ⅱ. ①弓… Ⅲ. ①Linux 操作系统－程序设计②C 语言－程序设计 Ⅳ.①TP316.89②TP312

中国版本图书馆 CIP 数据核字（2012）第 106596 号

策划编辑：胡辛征
责任编辑：李利键
特约编辑：赵树刚
印　　刷：北京东光印刷厂
装　　订：三河市皇庄路通装订厂
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×1092　1/16　印张：26.25　字数：672 千字
印　　次：2012 年 7 月第 1 次印刷
印　　数：4000 册　定价：59.80 元（含 DVD 光盘 1 张）

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

本 书 特 点

千里之行，始于足下！

——老子

为什么要写这样一本书

Linux 是一款非常优秀的开放源代码操作系统，在服务器解决方案方面和嵌入式领域有着巨大的优势。近年来，国内无数大中型企业选择 Linux 系统作为服务器解决方案，对服务器软件开发工作人员需求量巨大，但是相关的开发人员却非常紧缺。又因为 C 语言是 Linux 下最重要的编程语言，所以对 Linux 环境下的 C 程序开发人员的需求更为迫切。C 语言在语法结构和编程风格方面有其自身的特点，较其他编程语言难于学习和掌握。

基于上面的情况，笔者结合自身多年 Linux 下 C 程序开发经验，从一个初学者的角度系统地讲解 Linux 下 C 程序开发知识。书中对有关原理的阐述力求做到通俗易懂，并通过大量的编程实例进行讲解。在这本书中，笔者将通过 21 天的学习规划，让读者快速掌握 Linux 下 C 程序开发技能。

本书有何特色

相比同类图书，本书具有以下明显的特色。

1．细致体贴的讲解

为了让读者更快地上手，本书特别设计了适合初学者的学习方式，用准确的语言总结概念#用直观的图示演示过程#用详细的注释解释代码#用形象的比方帮助记忆。效果如下：

① 知识点介绍 准确、清晰是其显著特点，一般放在每一节开始位置，让零基础的读者了解相关概念，顺利入门。

② 范例 书中出现的完整实例，以章节顺序编号，便于检索和循序渐进地学习、实践，放在每节知识点介绍之后。

③ 范例代码 与范例编号对应，层次清楚、语句简洁、注释丰富，体现了代码优美的原则，有利于读者养成良好的代码编写习惯。对于大段程序，均在每行代码前设定编号，便于学习。

④ 运行结果 对范例给出运行结果和对应图示，帮助读者更直观地理解范例代码。

⑤ 代码解析 对范例代码中的关键代码进行逐一解释，有助于读者掌握相关概念和知识。

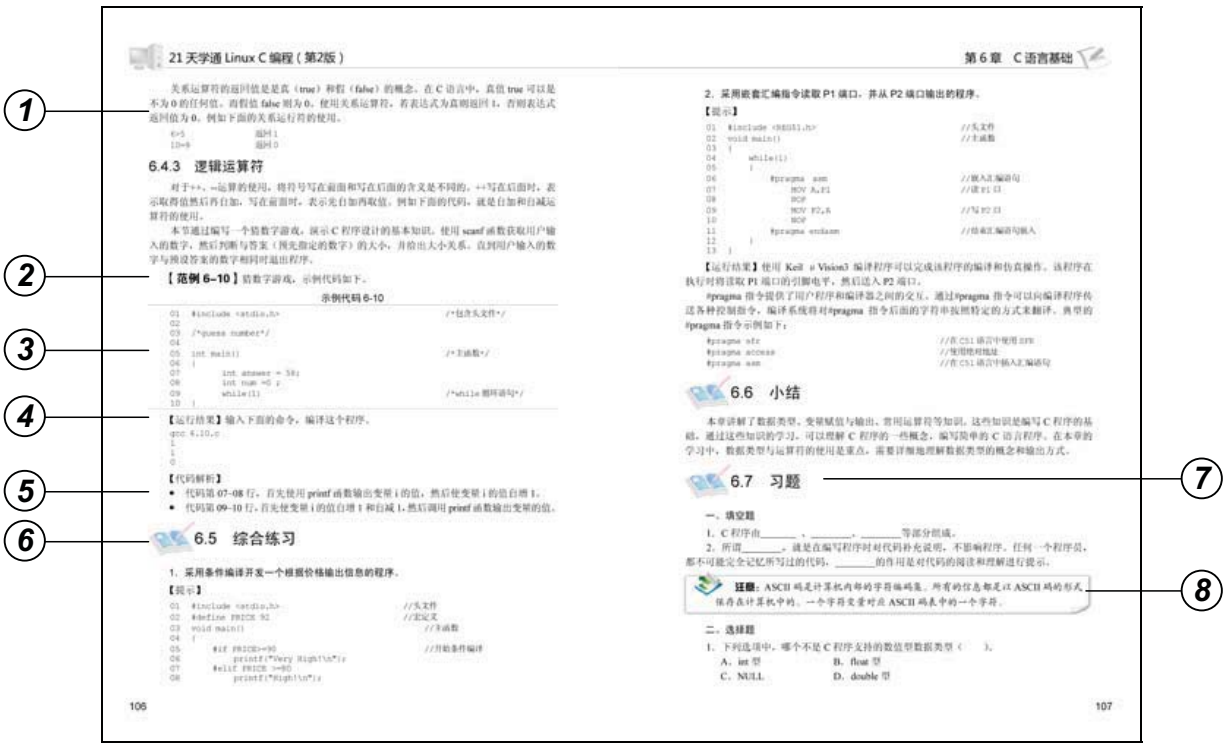
⑥ 综合练习 为了便于读者巩固所学内容，本书每章均提供了综合练习，并给出了操作提示和结果，配合读者自己动手实践。

⑦ 习题 每章最后提供专门的测试习题，供读者检验所学知识是否牢固掌握，题目的提示或答案放在光盘中。

⑧ **贴心的提示** 为了便于读者阅读，全书还穿插着一些技巧、提示等小贴士，体例约定如下。

- 提示：通常是一些贴心的提醒，让读者加深印象，或给读者提供建议及解决问题的方法。
- 注意：提出学习过程中需要特别注意的一些知识点和内容，或者相关信息。
- 警告：对操作不当或理解偏差将会造成的灾难性后果给出警示，以加深读者印象。

经作者多年的培训和授课证明，以上讲解方式是最适合初学者学习的方式，读者按照这种方式，会非常轻松、顺利地掌握本书知识。

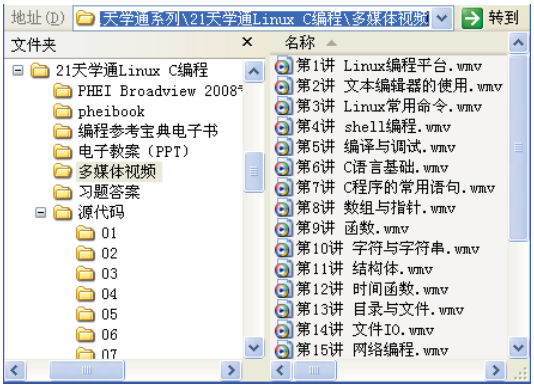


2．实用超值的 DVD 光盘

为了帮助读者比较直观地学习，本书附赠 DVD 光盘，内容包括多媒体视频、电子教案（PPT）、各章习题答案和实例源代码等。

- **多媒体视频**

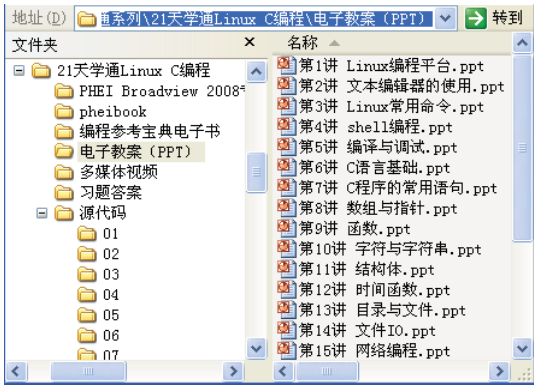
配有长达 21 小时手把手教学视频，讲解关键知识点界面操作和书中的一些综合练习题。作者亲自配音、演示，手把手教会读者使用。





● 电子教案（PPT）

本书可以作为高校相关课程的教材或课外辅导书，所以笔者特别为本书制作了电子教案（PPT），以便老师教学使用。



3．提供完善的技术支持

本书提供了论坛：http://www.rzchina.net，读者可以在上面提问交流。另外，论坛上还有一些小的教程、视频动画和各种技术文章，可帮助读者提高开发水平。

推荐的学习计划

本书从初学者角度出发，结合对学习阶段的认识，归纳了最适合初学者的学习模式，并为读者总结了合理的学习时间分配方式，列表如下：

时间安排		学习内容	教学目标	难度指数
第1周	第1天	熟悉并搭建 Linux 编程平台，了解常见的 Linux 发行版	了解常见的 Linux 发行版及各自的特点，掌握 Linux 操作系统的安装过程，能够使用虚拟机搭建 Linux 编程平台，能够在 Linux 下编写和编译 C 程序，并运行生成文件	★
	第2天	Linux 下文本编辑器的使用	熟练掌握 Linux 下的文本编辑器的使用，重点是掌握纯文本编辑器 vim 的使用，能够使用编辑器进行代码的快速编写	★★
	第3天	常见目录与文件操作命令的使用	熟练掌握各种常见的 Linux 目录和文件操作命令，掌握使用 man 命令和 info 命令获取命令帮助的方法并能够读懂帮助文档	★★
	第4天	Linux 下 bash 的使用，Shell 程序的基本结构	掌握 Linux 下 shell 编程的基本方法。能够利用 shell 特性和 Linux 的系统命令编程实现各种系统管理任务	★★★

续表

时间安排		学习内容	教学目标	难度指数
第 1 周	第 5 天	C 程序的编译概念，使用 gcc	理解编译的基本概念，能够使用 gcc 和 gdb 编译 C 程序和 C++程序，掌握 gdb 调试 C 程序的过程，能够通过 gdb 的帮助功能查阅常见 gdb 命令的用法，能够判断并纠正常见的编译错误	★★★
	第 6 天	C 程序的基本结构	了解 C 程序的基本结构，能够编写符合规范的简单的 C 程序，掌握 C 程序中变量的数据类型及变量的定义方法，能够为各种类型的变量赋值并输出变量的值，熟练掌握各种运算符的使用	★★★
	第 7 天	C 程序中的循环控制语句	掌握 C 程序中的各种流程控制语句的基本用法，在编程中能够选择合适的流程控制语句实现程序的功能，掌握问号运算符及块语句的用法	★★★
第 2 周	第 8 天	C 程序中的数组和指针	理解数组和指针的概念，掌握数组定义与数组元素的访问方法，掌握指针变量的定义方法，理解指针变量的算术运算的实质，能够使用指针变量的方式访问数组的元素，熟练掌握二维数组的定义与使用方法，了解多维数组	★★★★★
	第 9 天	函数的使用	理解函数的概念，能够使用系统的库函数完成程序的功能，能够自定义函数实现程序的概念，掌握函数的参数使用方法，能够利用指针实现函数参数的引用调用，熟悉各种常见的库函数及需要包含的头文件	★★★★
	第 10 天	字符串函数的使用	掌握各种字符与字符串处理函数的用法，能够使用这些函数灵活地实现字符与字符串的操作	★★★
	第 11 天	结构体的概念和使用	理解结构体的基本概念，能够定义结构体表示复杂的数据类型，掌握结构体数组及结构体指针的定义与访问方法，掌握在函数中使用结构体作为函数参数与返回值的用法	★★★★
	第 12 天	常用的时间函数	掌握各种获取系统时间的函数的用法，能够利用设置时间函数修改系统的时间	★★

续表

时间安排		学习内容	教学目标	难度指数
第 2 周	第 13 天	目录与文件的使用	理解 Linux 下文件的权限，了解常见的文件操作错误处理与错误号，能够利用系统函数实现文件和目录的创建与删除操作，掌握文件操作的整个流程，理解文件锁定的概念及用法，掌握实现文件的移动与复制的方法	★★★★★
	第 14 天	文件的输入和输出	掌握文件的打开与关闭函数的用法，掌握文件的读写函数的用法，能够利用文件操作函数实现常规的读写操作	★★★
第 3 周	第 15 天	网络编程的概念和应用	理解基本的网络概念，理解套接字，掌握套接字类型及相关的数据类型，掌握获取网络协议、网络服务及网络地址转换等相关信息的系统函数的用法，能够判断并处理网络编程中常见的错误	★★★★★
	第 16 天	无连接套接字通信的基本概念和创建方法	理解无连接套接字的基本概念及工作流程，掌握 socket 套接字的建立、状态获取和设置的方法，掌握无连接套接字发送信息与接收信息函数的用法，能够编写基本的面向无连接的网络服务器程序和网络客户端程序	★★★★★
	第 17 天	面向连接的套接字通信工作概念和流程	理解面向连接的套接字通信工作流程，熟练掌握绑定端口函数 bind、等待监听函数 listen、接受连接函数 accept、请求连接函数 connect、数据接收函数 recv、信息发送函数 send、数据传输函数 write 与 read，能够编写基本的面向连接的网络服务器程序和网络客户端程序	★★★★★
	第 18 天	Linux 的桌面环境	理解 Linux 下的 X Window 系统体系结构，了解 KDE 及 GNOME，掌握 GTK 开发库的安装，能够确定和纠正图形界面程序开发的常见错误	★★★
	第 19 天	Linux 图形界面编程	掌握窗口的创建及基本操作方法，了解图形界面编程中常见的界面元素，掌握窗口、标签、按钮及文本框的使用方法，能够利用这些元件编写基本的图形界面应用程序	★★★★

续表

时间安排		学习内容	教学目标	难度指数
第 3 周	第20天	界面布局的常用方法	熟练掌握使用表格进行界面布局的方法，会用垂直框和水平窗格进行界面布局，掌握界面布局工具 Glade 的使用方法，能够实现较为复杂的界面布局效果	★★★
	第21天	图形界面编程中信号与事件的基本概念和事件	理解信号与事件的基本概念和运行机制，掌握添加信号与事件的方法，能够编写基于事件驱动的图形界面应用程序	★★★★★

本书适合哪些读者阅读

- Linux 下 C 编程的自学人员和初学者；
- 高校计算机专业学生；
- 相关培训班的培训学员和老师；
- Linux 爱好者和工作人员；
- Linux 程序员；
- 其他编程爱好者。

编　著　者

目　录

第一篇　Linux 基础篇

第 1 章　Linux 编程平台（ 教学视频：64 分钟） ····· 23

1.1　Linux 发行版的介绍 ····· 23
1.1.1　Red Hat Linux ····· 23
1.1.2　Debian Linux ····· 23
1.2　安装 Linux ····· 24
1.2.1　Linux 的分区格式 ····· 24
1.2.2　在 Windows 系统中划分 Linux 分区 ····· 24
1.2.3　安装 Red Hat Linux 9.0 ····· 26
1.2.4　GRUB 在 Windows 分区中引导 Linux ····· 33
1.3　VMware 虚拟机安装 Linux 系统 ····· 34
1.3.1　VMware 软件安装 ····· 34
1.3.2　新建虚拟计算机 ····· 36
1.3.3　VMware 的使用 ····· 39
1.3.4　在 VMware 虚拟机中安装 Linux 系统 ····· 40
1.4　第一个 C 程序 ····· 41
1.4.1　在 Linux 中编写 C 程序代码 ····· 41
1.4.2　编译程序 ····· 42
1.5　小结 ····· 42
1.6　习题 ····· 43

第 2 章　使用文本编辑器（ 教学视频：39 分钟） ····· 44

2.1　使用 vim ····· 44
2.1.1　启动 vim ····· 44
2.1.2　在桌面上创建 vim 启动器 ····· 45
2.1.3　vim 的工作模式 ····· 45
2.1.4　保存与打开文件 ····· 45
2.1.5　移动光标 ····· 46
2.1.6　插入操作 ····· 48
2.1.7　删除操作 ····· 48
2.1.8　取消操作 ····· 49
2.1.9　保存操作 ····· 49
2.1.10　退出操作 ····· 50
2.1.11　查找操作 ····· 50
2.1.12　替换操作 ····· 50
2.1.13　选项设置 ····· 50
2.1.14　调用 shell 命令 ····· 51
2.2　vim 使用实例 ····· 51

2.2.1 字符的插入与删除……51
2.2.2 字符的查找与替换……52
2.3 gvim……53
2.3.1 文件的新建与保存……53
2.3.2 查找与替换……53
2.4 gedit……54
2.4.1 gedit 的启动与打开文本……54
2.4.2 gedit 编辑文件……55
2.4.3 gedit 打印文件……56
2.4.4 gedit 的首选项……57
2.5 小结……59
2.6 习题……59

第二篇 shell 编程篇

第 3 章 Linux 常用命令（ 教学视频：59 分钟）……60

3.1 目录与文件命令……60
3.1.1 查看当前路径命令 pwd……60
3.1.2 切换目录命令 cd……60
3.1.3 查看目录命令 ls……61
3.1.4 建立文件命令 touch……62
3.1.5 建立与删除目录命令 mkdir、rmdir……63
3.1.6 文件复制命令 cp……63
3.1.7 文件移动和更名命令 mv……64
3.1.8 删除文件或目录命令 rm……64
3.1.9 创建链接命令 ln……65
3.1.10 查看文件信息命令 wc……65
3.1.11 文件查看命令 cat……66
3.1.12 查看文件视图命令 head 与 tail……66
3.1.13 文件权限设置命令 chmod……67
3.1.14 查找文件命令 find……68
3.2 系统帮助命令 man……68
3.2.1 man 手册的使用方法……69
3.2.2 帮助文档的结构……69
3.3 小结……70
3.4 习题……70
3.5 面试题……70

第 4 章 shell 编程（ 教学视频：122 分钟）……72

4.1 bash 的使用……72
4.1.1 bash 的内部命令与外部命令……72
4.1.2 命令补齐功能……72
4.1.3 命令通配符……73
4.1.4 使用命令的历史记录……73

4.1.5 命令的别名 …… 74
4.2 shell 程序的基本结构 …… 74
4.3 局部变量 …… 75
4.4 环境变量 …… 76
4.4.1 查看环境变量 …… 76
4.4.2 访问环境变量 …… 76
4.4.3 定义环境变量 …… 76
4.4.4 在系统配置文件中定义环境变量 …… 77
4.5 位置变量 …… 77
4.6 shell 的运算符 …… 78
4.6.1 变量赋值 …… 78
4.6.2 算术运算符 …… 78
4.7 输入和输出 …… 80
4.7.1 echo 命令输出结果 …… 80
4.7.2 read 命令读取信息 …… 81
4.7.3 文件重定向 …… 82
4.8 引号的使用方法 …… 83
4.8.1 双引号 …… 83
4.8.2 单引号 …… 83
4.8.3 反引号 …… 84
4.8.4 反斜线 …… 84
4.9 测试语句 …… 84
4.9.1 文件状态测试 …… 85
4.9.2 数值测试 …… 85
4.9.3 字符串测试 …… 86
4.9.4 逻辑测试 …… 87
4.10 流程控制结构 …… 87
4.10.1 if 语句 …… 87
4.10.2 for 语句 …… 88
4.10.3 until 语句 …… 89
4.11 综合练习 …… 90
4.11.1 程序的功能 …… 90
4.11.2 编写程序的代码 …… 90
4.12 小结 …… 92
4.13 习题 …… 92
4.14 面试题 …… 92

第三篇 C 程序基础篇

第 5 章 编译与调试（ 教学视频：49 分钟） …… 93

5.1 编译的概念和理解 …… 93
5.1.1 程序编译的过程 …… 93
5.1.2 编译器 …… 93
5.2 gcc 编译器 …… 94

5.2.1 gcc 编译器简介 ……94
5.2.2 gcc 对源程序扩展名的支持……94
5.3 C 程序的编译 ……95
5.3.1 编写第一个 C 程序 ……95
5.3.2 用 gcc 编译程序 ……96
5.3.3 查看 gcc 的参数 ……97
5.3.4 设置输出的文件……98
5.3.5 查看编译过程……98
5.3.6 设置编译的语言……99
5.3.7 设置 ANSIC 标准……99
5.3.8 编译 C++程序……99
5.4 编译过程的控制……100
5.4.1 编译过程简介……100
5.4.2 控制预处理过程……101
5.4.3 生成汇编代码……101
5.4.4 生成目标代码……102
5.4.5 链接生成可执行文件……103
5.5 gdb 调试程序 ……103
5.5.1 gdb 简介……103
5.5.2 在程序中加入调试信息……103
5.5.3 启动 gdb……104
5.5.4 加载需要调试的程序……104
5.5.5 查看代码……104
5.5.6 在程序中加入断点……105
5.5.7 查看断点……105
5.5.8 运行程序……105
5.5.9 变量的查看……106
5.6 gdb 的帮助信息与常用命令……107
5.6.1 使用 gdb 的帮助命令……107
5.6.2 gdb 常用命令……107
5.7 编译程序常见的错误与问题……108
5.7.1 逻辑错误与语法错误……108
5.7.2 C 程序中的错误与异常 ……108
5.7.3 编译中的警告提示……109
5.7.4 找不到包含文件的错误……109
5.7.5 错误地使用逗号……109
5.7.6 括号不匹配错误……110
5.8 综合练习……110
5.8.1 编写程序……110
5.8.2 编译文件……111
5.8.3 调试程序……111
5.9 小结……113
5.10 习题 ……113
5.11 面试题 ……113

第 6 章　C 语言基础（ 教学视频：70 分钟） …… 114
6.1　C 程序的基本概念 …… 114
6.1.1　C 程序的基本结构 …… 114
6.1.2　C 程序的一般格式 …… 115
6.1.3　C 程序中的注释 …… 116
6.2　数据类型 …… 116
6.2.1　整型（int） …… 116
6.2.2　浮点型（float） …… 117
6.2.3　字符型（char） …… 118
6.2.4　变量名 …… 119
6.2.5　字符 NULL …… 120
6.3　变量的赋值与输出 …… 120
6.3.1　变量的赋值 …… 120
6.3.2　printf 函数输出变量 …… 120
6.3.3　scanf 函数从键盘读入变量 …… 122
6.4　运算符 …… 123
6.4.1　算术运算符 …… 123
6.4.2　关系运算符 …… 125
6.4.3　逻辑运算符 …… 125
6.5　综合练习 …… 126
6.6　小结 …… 127
6.7　习题 …… 127
6.8　面试题 …… 127
第 7 章　C 程序的常用语句（ 教学视频：87 分钟） …… 128
7.1　流程控制语句 …… 128
7.1.1　if 条件语句 …… 128
7.1.2　if 语句的嵌套 …… 130
7.1.3　switch 选择执行语句 …… 131
7.1.4　for 循环语句 …… 133
7.1.5　for 循环的嵌套 …… 135
7.1.6　for 循环应用实例：输出九九乘法口诀表 …… 136
7.1.7　while 循环语句 …… 137
7.1.8　do while 循环语句 …… 138
7.1.9　转移控制语句：continue …… 139
7.1.10　转移控制语句：break …… 140
7.1.11　转移控制语句：return …… 141
7.2　两种特殊语句结构 …… 141
7.2.1　三元操作符（?） …… 141
7.2.2　块语句 …… 142
7.3　综合练习 …… 143
7.4　小结 …… 144
7.5　习题 …… 144

7.6 面试题 ……144

第四篇 指针与函数篇

第 8 章 数组与指针（ 教学视频：66 分钟） ……146

8.1 数组的理解与操作 ……146
8.1.1 什么是数组 ……146
8.1.2 数组的定义与访问 ……147
8.1.3 数组使用实例 ……148
8.2 指针 ……149
8.2.1 指针的理解 ……149
8.2.2 指针操作符 ……149
8.2.3 指针的定义与访问 ……150
8.2.4 指针使用实例 ……151
8.3 数组与指针 ……151
8.3.1 数组与指针的关系 ……151
8.3.2 指针的算术运算 ……152
8.3.3 字符数组与字符串 ……153
8.4 二维数组与多维数组 ……154
8.4.1 二维数组的理解 ……154
8.4.2 二维数组与指针 ……156
8.5 实例 ……157
8.5.1 学生成绩统计实例 ……157
8.5.2 冒泡法排序实例 ……158
8.5.3 统计字符串中的字符 ……160
8.6 小结 ……162
8.7 习题 ……162
8.8 面试题 ……162

第 9 章 函数（ 教学视频：67 分钟） ……164

9.1 函数的理解 ……164
9.1.1 什么是函数 ……164
9.1.2 库函数与自定义函数 ……165
9.1.3 函数的返回值 ……166
9.1.4 无参函数 ……167
9.1.5 有参函数 ……167
9.1.6 函数参数实例 ……168
9.2 自定义函数 ……169
9.2.1 编写自定义函数 ……169
9.2.2 函数中调用函数 ……170
9.2.3 函数的声明 ……172
9.2.4 递归函数 ……173
9.2.5 main 函数的参数 argc 与 argv ……174
9.3 函数与指针 ……175

9.3.1 值调用与引用调用 …… 175
9.3.2 引用调用与指针 …… 176
9.3.3 指针参数简单实例 …… 177
9.3.4 自定义函数中指针使用实例 …… 178
9.3.5 数组作为参数 …… 179
9.3.6 数组作为函数参数实例 …… 180
9.4 返回值 …… 182
9.4.1 函数返回值的类型 …… 182
9.4.2 函数返回指针 …… 182
9.4.3 函数返回指针实例 …… 183
9.4.4 函数返回结果到参数 …… 184
9.5 综合练习 …… 185
9.6 小结 …… 186
9.7 习题 …… 186
9.8 面试题 …… 186

第 10 章 字符与字符串处理（ 教学视频：66 分钟） …… 188

10.1 字符测试函数 …… 188
10.1.1 数字或字母测试函数 isalnum …… 188
10.1.2 字母测试函数 isalpha …… 189
10.1.3 可打印字符测试函数 isgraph …… 190
10.1.4 测试大小写字母函数 islower 和 isupper …… 191
10.1.5 测试是否是数字 isxdigit …… 192
10.1.6 测试字符是否为符号函数 ispunct …… 192
10.1.7 其他字符测试函数 …… 193
10.2 字符测试函数综合实例 …… 194
10.3 字符串转换 …… 195
10.3.1 C 程序中的字符串 …… 195
10.3.2 字符串转换成浮点型函数 atof …… 196
10.3.3 字符串转换成整型函数 atoi …… 196
10.3.4 字符串转换成长整型函数 atol …… 197
10.3.5 将浮点型转换成字符串函数 ecvt …… 198
10.3.6 字母的大写小转换函数 tolower 和 toupper …… 199
10.3.7 其他字符串转换函数 …… 200
10.4 字符串比较 …… 201
10.4.1 字符串比较函数 bcmp …… 201
10.4.2 字符串大小比较函数 memcmp …… 202
10.4.3 忽略大小写比较字符串函数 strncasecmp …… 203
10.5 字符串复制 …… 204
10.5.1 字符串复制函数 bcopy …… 204
10.5.2 字符串复制函数 memccpy …… 204
10.5.3 复制字符串函数 strcpy …… 205
10.5.4 字符串复制函数 strncpy …… 206
10.6 字符串的清理与填充 …… 207

10.6.1 字符串清理函数 bzero……207
10.6.2 字符串填充函数 memset……208
10.7 字符串查找……208
10.7.1 字符查找函数 index 与 rindex……208
10.7.2 字符查找函数 memchr……209
10.7.3 字符查找函数 strchr 与 strrchr……210
10.8 字符串的连接与分割……211
10.8.1 字符串连接函数 strcat……211
10.8.2 字符串分割函数 strtok……211
10.9 综合练习……212
10.10 小结……214
10.11 习题……215
10.12 面试题……215

第 11 章 结构体（ 教学视频：49 分钟）……216

11.1 结构体的操作……216
11.1.1 结构体的理解……216
11.1.2 结构体的定义……216
11.1.3 结构体的访问……218
11.1.4 结构体数组……220
11.1.5 结构体使用实例……220
11.2 结构体与指针……223
11.2.1 结构体指针的定义……223
11.2.2 结构体指针的访问……223
11.2.3 结构体作为函数的参数……225
11.2.4 结构体指针作为函数的参数……226
11.2.5 结构体作为函数的返回值……228
11.2.6 结构体指针作为函数的返回值……230
11.3 综合练习……231
11.3.1 程序的需求分析……231
11.3.2 程序中的函数……232
11.3.3 程序中的结构体与全局变量……232
11.3.4 头文件和函数声明……233
11.3.5 定义结构体和全局变量……233
11.3.6 显示学生信息的函数……233
11.3.7 程序的选择菜单……234
11.3.8 学生信息输入函数……235
11.3.9 按姓名查找函数……235
11.3.10 删除学生函数……236
11.3.11 按年龄查找函数……237
11.3.12 按身高查找函数……237
11.3.13 按性别查找函数……238
11.3.14 程序的主函数……238
11.3.15 程序的运行和调试……240
11.4 小结……241

11.5 习题 …… 242
11.6 面试题 …… 242

第 12 章　时间函数（ 教学视频：23 分钟）……243
12.1　常用时间函数……243
12.1.1　返回时间函数 time……243
12.1.2　取当前时间函数 gmtime……244
12.1.3　字符串格式时间函数 ctime……246
12.1.4　字符串格式时间函数 asctime……247
12.1.5　取得当地时间函数 localtime……248
12.1.6　将时间转换成秒数函数 mktime……249
12.1.7　取得目前的时间函数 gettimeofday……250
12.1.8　设置当前时间函数 settimeofday……251
12.2　综合练习……252
12.3　小结……253
12.4　习题……253
12.5　面试题……253

第五篇　文件与目录篇

第 13 章　目录与文件（ 教学视频：115 分钟）……255
13.1　文件操作的权限……255
13.2　错误处理与错误号……256
13.2.1　错误定义的理解……256
13.2.2　用错误常数显示错误信息……257
13.2.3　用错误序号显示错误信息……258
13.3　创建与删除目录……258
13.3.1　创建目录函数 mkdir……259
13.3.2　删除目录函数 rmdir……260
13.4　文件的创建与删除……261
13.4.1　创建文件函数 creat……261
13.4.2　删除文件函数 remove……262
14.4.3　建立临时文件函数 mkstemp……263
13.5　文件的打开与关闭……265
13.5.1　打开文件函数 open……265
13.5.2　文件打开方式的设置……265
13.5.3　打开文件的权限……265
13.5.4　文件打开实例……265
13.5.5　关闭文件函数 close……267
13.6　文件读写……268
13.6.1　写文件函数 write……268
13.6.2　读取文件函数 read……270
13.6.3　文件读写位置的移动……271
13.6.4　将缓冲区数据同步到磁盘的函数 sync……273
13.6.5　将指定文件的缓冲区数据同步到磁盘的函数 fsync……274

13.7 文件锁定 …… 275
13.7.1 文件锁定的理解 …… 276
13.7.2 文件锁定函数 flock …… 276
13.7.3 文件锁定函数 fcntl …… 278
13.7.4 文件锁定函数 fcntl 使用实例 …… 279
13.8 文件的移动与复制 …… 280
13.8.1 文件的移动函数 rename …… 280
13.8.2 rename 函数使用实例 …… 280
13.8.3 文件复制实例 …… 282
13.9 综合练习 …… 283
13.9.1 程序功能分析 …… 283
13.9.2 程序的函数 …… 283
13.9.3 包含文件 …… 284
13.9.4 数据的定义 …… 284
13.9.5 菜单函数 …… 284
13.9.6 显示电话信息函数 …… 286
13.9.7 添加电话号码函数 …… 286
13.9.8 按姓名查找函数 …… 286
13.9.9 删除电话号码函数 …… 287
13.9.10 保存到文件函数 …… 288
13.9.11 从文件导入信息函数 …… 288
13.9.12 主函数 …… 289
13.10 小结 …… 290
13.11 习题 …… 291
13.12 面试题 …… 291

第 14 章 文件 I/O（ 教学视频：42 分钟） …… 292

14.1 文件的打开与关闭 …… 292
14.1.1 文件打开函数 fopen …… 292
14.1.2 文件打开函数 fdopen …… 294
14.1.3 打开文件函数 freopen …… 295
14.1.4 关闭文件函数 fclose …… 296
14.2 文件的读写 …… 297
14.2.1 字符写入函数 putc 与 fputc …… 297
14.2.2 向文件中写入字符串函数 fputs …… 298
14.2.3 数据写入函数 fwrite …… 299
14.3 综合练习 …… 300
14.4 小结 …… 301
14.5 习题 …… 301
14.6 面试题 …… 301

第六篇 网络编程篇

第 15 章 网络编程（ 教学视频：71 分钟） …… 302

15.1 网络编程的基本概念……302
15.1.1 IP 地址……302
15.1.2 端口……303
15.1.3 域名……303
15.1.4 TCP 与 UDP……303
15.2 套接字……304
15.2.1 什么是套接字……304
15.2.2 套接字相关的数据类型……304
15.2.3 套接字类型……305
15.3 域名与 IP 地址……305
15.3.1 用域名取得主机的 IP 地址……305
15.3.2 用 IP 地址返回域名……307
15.4 网络协议……308
15.4.1 由协议名取得协议数据……308
15.4.2 由协议编号取得协议信息……310
15.4.3 取得系统支持的所有协议……311
15.5 网络服务……312
15.5.1 取得系统支持的网络服务……312
15.5.2 用名称取得系统所支持的服务……313
15.5.3 由端口取得服务名称……315
15.6 网络地址的转换……316
15.6.1 将网络地址转换成整型……316
15.6.2 将长整型数地址转换成网络地址……317
15.6.3 主机字符顺序与网络字符顺序的转换……318
15.7 错误处理……318
15.7.1 herror 函数显示错误……318
15.7.2 捕获错误编号……319
15.8 综合练习……320
15.9 小结……321
15.10 习题……321
15.11 面试题……321

第 16 章 无连接的套接字通信（ 教学视频：22 分钟）……322

16.1 socket 套接字……322
16.1.1 建立 socket……322
16.1.2 取得 socket 状态……324
16.1.3 设置 socket 状态……325
16.2 无连接的套接字通信介绍……326
16.2.1 工作流程……327
16.2.2 信息发送函数 sendto……327
16.2.3 信息接收函数 recvfrom……328
16.3 综合练习……328
16.3.1 无连接套接字通信客户端……328
16.3.2 无连接套接字通信服务器……330

16.3.3 测试 UDP 通信程序 …… 331
16.4 小结 …… 332
16.5 习题 …… 332
16.6 面试题 …… 333

第 17 章 面向连接的套接字通信（ 教学视频：76 分钟） …… 334

17.1 面向连接的套接字通信工作流程 …… 334
17.2 绑定端口 …… 335
17.2.1 绑定端口函数 bind …… 335
17.2.2 bind 函数绑定端口实例 …… 336
17.3 监听与连接 …… 337
17.3.1 等待监听函数 listen …… 337
17.3.2 listen 函数使用实例 …… 337
17.3.3 接受连接函数 accept …… 339
17.3.4 accept 函数使用实例 …… 339
17.3.5 请求连接函数 connect …… 341
17.3.6 connect 函数使用实例 …… 341
17.4 数据的发送与接收 …… 343
17.4.1 数据接收函数 recv …… 343
17.4.2 recv 函数使用实例 …… 343
17.4.3 信息发送函数 send …… 345
17.4.4 数据传输函数 write 与 read …… 345
17.4.5 read 函数接收数据实例 …… 346
17.5 综合练习 …… 348
17.5.1 服务器程序 …… 348
17.5.2 客户端程序 …… 350
17.5.3 实例程序测试 …… 351
17.6 小结 …… 352
17.7 习题 …… 352
17.8 面试题 …… 353

第七篇 图形界面开发篇

第 18 章 Linux 图形界面开发基础（ 教学视频：28 分钟） …… 354

18.1 X Window 系统简介 …… 354
18.1.1 X Window 的历史 …… 354
18.1.2 X Window 体系结构 …… 355
18.2 Linux 桌面环境 …… 355
18.2.1 KDE …… 355
18.2.2 GNOME …… 356
18.3 Linux 下的图形程序开发环境 …… 357
18.3.1 GTK+简介 …… 357
18.3.2 yum 安装 GTK …… 357
18.3.3 用软件包管理器安装 GTK …… 358

18.4 常见问题……359
18.4.1 找不到 gtk.h 包含文件的错误……359
18.4.2 pkg-config 参数有什么作用……359
18.5 综合练习……359
18.5.1 编写程序代码……359
18.5.2 编译程序……360
18.5.3 运行程序……361
18.6 小结……361
18.7 习题……361
18.8 面试题……362

第 19 章 图形界面编程（ 教学视频：35 分钟）……363

19.1 窗口……363
19.1.1 最简单的窗口程序……363
19.1.2 窗口的构建过程……364
19.1.3 新建窗口函数 gtk_window_new……364
19.1.4 设置窗口的标题……365
19.1.5 设置窗口的大小与位置……366
19.2 标签……367
19.2.1 建立一个标签……367
19.2.2 将标签添加到窗口中……367
19.2.3 设置与获取标签的文本……368
19.3 按钮……369
19.3.1 添加一个按钮……370
19.3.2 取得与设置按钮的标签……371
19.4 文本框……372
19.4.1 建立文本框……372
19.4.2 获取与设置文本框数据……373
19.5 综合练习……374
19.6 小结……375
19.7 习题……375
19.8 面试题……376

第 20 章 界面布局（ 教学视频：67 分钟）……377

20.1 表格……377
20.1.1 表格的建立……377
20.1.2 表格中添加元件……378
20.1.3 表格使用实例……379
20.1.4 单元格的合并……380
20.1.5 表格中添加表格……381
20.2 垂直框……383
20.2.1 添加垂直框……383
20.2.2 垂直框中添加元件……384
20.3 水平窗格……385

20.3.1 新建一个窗格 …… 385
20.3.2 窗格中添加元件 …… 387
20.4 Glade …… 388
20.4.1 Glade 简介 …… 388
20.4.2 使用 Glade 设计程序界面 …… 388
20.4.3 Glade 设计窗口 …… 389
20.4.4 添加表格 …… 392
20.4.5 添加标签 …… 392
20.4.6 添加文本框 …… 395
20.4.7 添加按钮 …… 396
20.4.8 添加信号与事件 …… 397
20.4.9 Glade 实例 …… 398
20.5 综合练习 …… 399
20.6 小结 …… 400
20.7 习题 …… 401
20.8 面试题 …… 401

第 21 章 信号与事件（ 教学视频：18 分钟） …… 402

21.1 信号 …… 402
21.1.1 添加信号 …… 402
21.1.2 事件函数的参数 …… 403
21.1.3 在函数中使用全局变量 …… 404
21.2 GTK 事件 …… 406
21.3 实例：加法计算器 …… 406
21.3.1 程序功能的分析 …… 406
21.3.2 程序实现 …… 406
21.4 综合练习 …… 408
21.4.1 程序的功能分析 …… 408
21.4.2 程序的功能模块 …… 409
21.4.3 包含头文件与定义全局变量 …… 410
21.4.4 单击按钮输入函数 …… 410
21.4.5 加法运算函数 …… 410
21.4.6 减法运算函数 …… 411
21.4.7 乘法运算函数 …… 412
21.4.8 除法运算函数 …… 412
21.4.9 输入小数点函数 …… 413
21.4.10 清除数据函数 …… 413
21.4.11 界面构建函数 …… 413
21.4.12 添加事件函数 …… 415
21.4.13 主函数 …… 416
21.4.14 程序的编译和运行 …… 416
21.5 小结 …… 417
21.6 习题 …… 417
21.7 面试题 …… 417

第 4 章　shell 编程

shell 是 Linux 操作系统的命令解释程序，它负责将用户输入的命令传递给内核执行并将执行结果反馈给用户。熟练掌握 shell 的使用将提高用户在命令行中的工作效率。利用 shell 程序还可以编写 shell 脚本，把经常性的工作转换为 shell 程序，方便以后使用。本章介绍 Linux 下的默认 shell 程序 bash 的使用及脚本编程，主要内容如下：

- shell 的基本使用；
- shell 程序的基本结构；
- shell 中的局部变量和环境变量；
- shell 中的运算符；
- shell 中的输入与输出；
- shell 中的测试语句与流程控制语句。



4.1　bash 的使用

在 Linux 操作系统中有多种 shell 可供使用，在 Red Hat Linux 9.0 中默认使用的 shell 是 bash。用户成功登录系统后，shell 将成为用户与系统内核进行交互的桥梁，直至用户退出系统。Linux 系统中所有用户都有一个默认的 shell。用户的默认 shell 在系统文件/etc/passwd 中指定。

4.1.1　bash 的内部命令与外部命令

Linux 的命令可以分为内部命令和外部命令。内部命令在 shell 启动时就调入内存，是常驻内存的。而外部命令是系统的软件功能，用户需要时才从硬盘中读入内存。例如下面的命令就是几个常用的内部命令。

- exit：终止当前 shell 的执行。
- export：设置一个环境变量，当前 shell 的所有子进程都可以访问此环境变量。
- kill：终止某个进程的执行。需要传递进程 PID 参数，终止对应进程的执行。



提示：外部命令的查找路径由环境变量 PATH 定义。

4.1.2　命令补齐功能

命令补齐可以自动补齐没有输入完整的命令。当用户不能拼写整个命令时，只需要输入开头的几个字符，然后按下“Tab”键即可调用 bash 的命令补齐功能。如果前面几个字符输入没有错误，并且以这几个字符开头的命令只有一个，系统会自动补齐整个命令。如果以这几个字符开头的目录不唯一，则再次按下“Tab”键时，会显示所有以这几个字符开头的命令。命令补齐不但能够对命令输入进行提示，还可以加快命令输入的速度。例如下面的操作使用了 bash 的命令补齐功能。



① 单击“主菜单”|“系统工具”|“终端”命令，打开系统终端。

② 在终端中输入“ifco”，然后按“Tab”键，这时会自动补齐为“ifconfig”命令。

③ 在终端中输入“ch”，然后按两次“Tab”键，这时会列出所有以 ch 开头的命令，这些命令如下所示。这时可以使用这些提示书写相关命令。

```
chacl    cheatmake    chinput      chmoddic     chroot
chage    checkXML     chkconfig    chooser      chsh
chat     chfn         chkfontpath  chown        chvt
chattr   chgrp        chmod        chpasswd
```

4.1.3　命令通配符

所谓通配符，就是指可以在命令中用一个字符来代替一系列字符或字符串。bash 中有三种通配符，其中?和[]代表单个字符。*可以代表一个或多个字符，也可以是空字符串。

- *：匹配任何字符和字符串，包括空字符串。
- ?：匹配任意一个字符。例如，?abc，可以匹配任何以 abc 结束，并且以任意字符开头的含有四个字符的字符串。
- [...]：匹配括号里列出的任何单字符。比如 abc[def]，可以匹配以 abc 开头，以 def 中任意一个字符结尾的字符串。

例如下面的例子在命令中使用通配符。

① 从主菜单中打开一个终端。

② 查看主目录下所有的 C 程序文件。C 程序文件的扩展名都是.c。所以输入以下的命令。

```
ls *.c                                  注意：用*匹配一个任意字符串。
```

命令的运行结果如下所示。

```
01.c  02.c  03.c  aa.c  h.c  hello.c
```

③ 列出用户主目录中以 0 开头的 C 程序文件，输入的命令如下所示。

```
ls 0*.c                                 注意：用*匹配一个任意字符串。
```

命令的运行结果如下所示。

```
01.c  02.c  03.c
```

④ 列出用户主目录中文件名只有 2 个字符的 C 程序文件。输入的命令如下所示。

```
ls ??.c                                 注意：用?匹配一个单一字符。
```

命令的运行结果如下所示。

```
01.c  02.c  03.c  aa.c
```

4.1.4　使用命令的历史记录

在终端中，如果需要再次使用已经输入过的命令，按向上方向键可以依次显示以前的命令。查找到需要的命令以后，按“Enter”键即可执行这个命令。用此方法，可以向前查找已经输入过的命令。history 命令可以显示所有的命令记录列表，命令的用法如下所示。

```
history [n]
```

参数 *n* 是一个可选的整数。当没有参数时，会列出以前执行过的所有命令。有参数 *n* 时，会列出最后执行的 *n* 个命令。例如用下面的命令来查看已经执行过的操作。

```
history 5
```

这个命令会显示最后 5 个命令的操作列表，结果如下所示。

```
152  ls ??.c
```

```
153  fc
154  ls ??.c
155  ls
156  history 5
```

4.1.5 命令的别名

命令别名指的是自定义命令代替其他命令，可以作为其他命令的缩写，用来减少键盘输入。别名的定义命令为 alias，用法如下所示。

```
alias list='ls -l'                    注释：定义一个文件列表的别名。
alias allfile='ls -a'                 注释：定义显示所有文件命令的别名。
alias lsc='ls *.c'                    注释：定义显示所有 c 程序文件命令的别名。
```



注意：定义别名时，等号的两边不能有空格，命令别名使用单引号括起来。

如果想取消别名，可以使用 unalias 命令。例如下面的命令可取消上面的代码定义的 lsc 别名。

```
unalias lsc
```



4.2 shell 程序的基本结构

shell 程序就是指将一系列的 Linux 程序通过流程控制写在一个文本文件中，由 shell 读取此文本文件并且依次执行文件中的这些程序。本节将以一个简单的 shell 程序为例讲解 shell 程序的结构。

① 打开终端，在终端中输入 vim 命令，按“Enter”键进入 vim。

② 在 vim 中按“i”键进入插入模式。然后输出下面的文本。

```
01  #!/bin/bash
02  #hello                          #注意：#后面的内容是 shell 程序的注释。
03  echo 'hello Linux'              #打印字符串到终端
04  echo 'this is a shell file.'
```



注意：在 shell 程序中，“ #”后边的内容为注释信息，不影响程序的执行。

③ 在 vim 中，按“Esc”键进入命令模式。然后输入命令:w a.sh，保存这个文件到用户主目录下，文件名为 a.sh。

④ 输入命令:q，退出 vim。

⑤ 输入下面的命令对文件 a.sh 添加可执行的权限。文本文件是没有执行权限的。

```
chmod +x a.sh
```



注意：在 Linux 系统中，不论是 shell 程序还是二进制程序，都必须具有可执行权限才能够运行。

⑥ 输入下面的命令运行这个 shell 程序。这个程序执行了两次字符串输出。

```
./a.sh
```

⑦ 程序的运行结果如下所示。

```
hello Linux
```

```
this is a shell file.
```

本例中的程序虽然简单，但包含了 shell 程序的下面一些基本特征：

① 所有 shell 程序的第一行都是以#!开头。后面跟上执行此 shell 程序的 shell 解释器完整路径。Red Hat Linux 9.0 的默认 shell 解释器是 bash。本书中所有的 shell 程序都使用 bash 来解释执行。

② 程序的第 2 行以注释的方式写出程序的名称，这是 shell 编程的一种习惯。

③ 最简单的 shell 程序就是一组 shell 命令的堆砌。在这个程序中，使用两个 echo 命令来输出两个字符串。

④ shell 程序是一个普通的文本文件，需要添加可执行权限以后才可以执行。如果试图执行一个没有可执行权限的 shell 程序，得到的输出结果如下所示。

```
bash: ./c.sh 权限不够
```



4.3　局部变量

shell 程序中需要用变量来存储程序的数据。shell 中的变量可分为局部变量、环境变量、位置变量三种。本将讲述 shell 程序中变量的使用方法。shell 语言是一种解释型语言，它与 C 或 Java 语言不同，编程时无须事先声明变量。当对一个变量进行赋值时，也就定义了变量。局部变量指的是只在当前的进程和程序中有效的变量。

shell 程序的变量是无数据类型的，可以使用同一个变量存放不同数据类型的值。变量赋值之后，只需在变量前面加一个$符号即可访问变量的值。可以用赋值符号（=）为变量赋值。变量的值如果是不带空格的字符串，则可以不加引号。变量输出使用的命令是 echo。例如下面的步骤建立一个 shell 程序，在程序中定义变量并输出变量的值。

【范例 4-1】shell 脚本中变量的定义与赋值，实例代码如下所示。

实例代码 4-1

```
01  #!/bin/bash
02  #bianliang 4.1.sh
03  a=123                          #直接为变量赋值
04  b=1.23
05  c=xyz
06  d=efgh xyz
07  e='efgh xyz'
08  echo $a                        #打印变量的值
09  echo $b
10  echo $c
11  echo $d
12  echo $e
```

【运行结果】在终端中输入下面的命令：

```
chmod +x 4.1.sh
```

在终端中输入命令“./4.1.sh”，然后按“Enter”键运行这个 shell 程序。程序的运行结果如下：

```
123                            $a 是一个整数赋值。
1.23                           $b 是一个小数赋值。
xyz                            $c 是一个字符串赋值。
                               $d 赋值时出现空格，赋值错误。
efgh xyz                       $e 用引号将一个含空格的字符串引起来再赋值。
```

【代码解析】

- 代码第 03~07 行在赋值时完成变量的定义。
- 代码第 08~12 行依次打印变量的值。



提示：在本章后面的例子中，如果没有特殊说明，都将在 vim 中输入代码。



4.4 环境变量

环境变量是在一个用户的所有进程中都可以访问的变量。系统中常常使用环境变量来存储常用的信息。本节将讲述环境变量的查看、访问和定义等操作。

4.4.1 查看环境变量

使用 export 命令可以查看系统的环境变量列表。打开一个终端，在一个终端中输入命令 export，然后按“Enter”键。在终端中显示的环境变量列表如下所示。

```
declare -x BASH_ENV="/root/.bashrc"
declare -x COLORTERM="gnome-terminal"
declare -x DESKTOP_STARTUP_ID=""
declare -x DISPLAY=":0.0"
declare -x GDMSESSION="Default"
declare -x GNOME_DESKTOP_SESSION_ID="Default"
declare -x GTK_RC_FILES="/etc/gtk/gtkrc:/root/.gtkrc-1.2-gnome2"
declare -x G_BROKEN_FILENAMES="1"
declare -x HISTSIZE="1000"
declare -x SHELL="/bin/bash"
declare -x SHLVL="2"
declare -x SSH_AGENT_PID="1858"
declare -x SSH_ASKPASS="/usr/libexec/openssh/gnome-ssh-askpass"
declare -x SSH_AUTH_SOCK="/tmp/ssh-XXEMD4KZ/agent.1804"
declare -x TERM="xterm"
declare -x USER="root"
declare -x USERNAME="root"
declare -x WINDOWID="29512886"
declare -x XAUTHORITY="/root/.Xauthority"
declare -x XMODIFIERS="@im=Chinput"
```

4.4.2 访问环境变量

环境变量的访问，需要在一个环境变量的前面加一个$符号。例如在终端中执行下面的命令可以输出环境变量的值。

```
echo $SSH_ASKPASS
```

程序的运行结果如下所示。

```
/usr/libexec/openssh/gnome-ssh-askpass
```

4.4.3 定义环境变量

可以用 export 命令来定义一个环境变量。环境变量的命令一般都是大写的。在终端中输入下面的命令来定义一个环境变量。

```
export XX=1234
```

可以用下面的语句来访问输出这个环境变量。

```
echo $XX
```

命令的运行结果如下所示。

```
1234
```

4.4.4　在系统配置文件中定义环境变量

上一节定义的环境变量，只在当前 shell 中运行的所有进程中有效，并没有保存到系统文件中。所以系统重启后就没法再访问这些环境变量。为了实现重启系统后，仍然能够访问定义的环境变量，可以在系统配置中加入这些环境变量的定义。环境变量的系统配置文件是/etc/profile。用下面的步骤在系统文件中查看和配置环境变量。

① 打开一个终端。在终端中输入并执行下面的命令。

```
gedit /etc/profile
```

② 文本编辑器会打开系统环境变量配置文件/etc/profile。可以在文件中发现，有下面这样的语句输出了环境变量。

```
export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE INPUTRC
```

③ 在文件的最后一行输入下面的代码，添加两个环境变量。

```
export A1=hello
export A2=12345
```

④ 单击文本编辑器的“文件”|“保存”菜单命令，保存这个文件。

⑤ 重新启动计算机以后，系统中会产生上面所设置的两个环境变量。



4.5　位置变量

位置变量指的是 shell 程序在运行时传入的参数。程序中可以用变量的形式来调用这些参数。这些参数被存放在 1~9 的 9 个特殊变量名中，被形象地称为位置变量。同普通变量一样，位置变量用$前缀加数字来表示。例如，第 5 个参数，表示为$5。例如要向 shell 程序传递参数“Beijing is a beautiful city”，用表 4.1 来说明如何访问每一个参数。

表 4.1　shell 中的位置变量

$0	$1	$2	$3	$4	$5	$6	$7	$8	$9
bash	Beijing	is	a	Beautiful	city				

在位置变量中，$0 的值为 bash。$1 以后的变量是输入参数的列表。例如下面的 shell 程序是访问位置变量的实例。

【范例 4-2】输出 shell 中的位置变量，实例代码如下所示。

实例代码 4-2

```
01  #!/bin/bash
02  #4.2.sh;
03  echo $1
04  echo $2
05  echo $3
06  echo $4
```

【运行结果】输入下面的命令对这个文件添加可执行的权限。

```
chmod +x 4.2.sh
```

输入下面的命令执行这个程序，在命令中输入程序的参数。

```
./4.2.sh Beijing is a beautiful city
```

程序输出了$1 到$4 位置变量中的 4 个参数，运行结果如下所示。

```
Beijing
is
a
beautiful
```

【代码解析】这个程序的内容很简单，用 echo 语句输出 shell 运行时输入的 4 个参数。



4.6 shell 的运算符

shell 中的运算符可以实现变量的赋值、算术运算、测试、比较等功能，运算符是构成表达式的基础。本章将讲述 shell 中运算符的使用。

4.6.1 变量赋值

shell 中使用“=”进行变量赋值，也可以用等号来改变或初值化一个变量的值。在进行赋值时不用考虑变量的数据类型，这是由 shell 中变量数据类型的特点决定的。例如下面是使用等号进行赋值的例子。

① 从主菜单中打开一个终端。

② 在终端中输入下面的命令对一个变量进行赋值。

```
STR=123
```

③ 输入下面的命令输出这一个变量的值。

```
echo $STR
```

④ shell 会输出这一个变量的值，结果如下所示。

```
123
```

⑤ 可以对同一个变量赋予不同的值，命令如下所示。

```
STR=asdf
```

⑥ 输入下面的变量输出变量的值。

```
echo $STR
```

⑦ 变量值的输出结果如下所示。

```
asdf
```

4.6.2 算术运算符

算术运算符指的是可以在程序中实现加、减、乘、除等数学运算的运算符。shell 中常用的数学运算符如下所示。

- +：对两个变量做加法。
- -：对两个变量做减法。
- *：对两个变量做乘法。
- /：对两个变量做除法。
- **：对两个变量做幂运算。
- %：取模运算，第一个变量除以第二个变量求余数。
- +=：加等于，在自身基础上加第二个变量。
- -=：减等于，在第一个变量的基础上减去第二个变量。
- *=：乘等于，在第一个变量的基础上乘以第二个变量。

- /=：除等于，在第一个变量的基础上除以第二个变量。
- %=：取模赋值，第一个变量对第二个变量取模运算，再赋值给第一个变量。

在使用这些运算符时，需要注意到运算顺序的问题。例如输入下面的命令，输出 1+2 的结果。

```
echo 1+2
```

shell 并没有输出结果 3，而是输出了 1+2。在 shell 中有 3 种方法可以更改运算顺序。

- 用 expr 改变运算顺序。可以用 echo `expr 1 + 2`来输出 1+2 的结果，用 expr 表示后面的表达式为一个数学运算。需要注意的是，`并不是一个单引号，而是“Tab”键上面的那个符号。
- 用 let 指示数学运算。可以先将运算的结果赋值给变量 b，运算命令是 let b=1+2。然后用 echo $b 来输出 b 的值。如果没有 let，则会输出 1+2。



注意：在“ let b=1+2” 中，加号的前后不能有空格。在`expr 1 + 2`表达式中，加号前后都需要有空格。

- 用$[]表示数学运算。将一个数学运算写到$[]符号的中括号中，中括号中的内容将先进行数学运算。例如命令 echo $[1+2]，将输出结果 3。

范例 4-3 所示是一个shell程序实例，实现数学函数$S=3(x^y)+4x^2+5y+6$ 的运算。在程序中以位置变量的方式输入x与y的值。程序的编写步骤如下所示。

【范例 4-3】用 shell 脚本进行数学运算实例，实例代码如下所示。

实例代码 4-3

```
01  #!/bin/bash
02  #4.3.sh
03  s=0                                  #定义一个求和变量，初值为 0。
04  t=`expr $1**$2`                      #用 expr 改变运算顺序，求 x 的 y 次方。
05  t=$[t*3]                             #t 乘以 3。
06  s=$[s+t]                             #结果相加。
07  t=$[$1**2]                           #求 x 的平方。
08  t=$[t*4]                             #结果乘以 4。
09  s=$[s+t]                             #结果相加。
10  t=`expr $2*5`                        #求 5y 的值。
11  s=$[s+t]                             #结果相加。
12  s=$[s+6]                             #结果加上 6。
13  echo $s                              #输出结果。
```

【运行结果】在终端中，输入下面的命令为 4.3.sh 文件添加可执行权限。

```
chmod +x 4.3.sh
```

输入下面的命令运行程序。在命令中需要输入两个参数。

```
./4.3.sh 2 4
```

结果如下所示。

```
90
```

【代码解析】

- 代码第 04~10 行代码首先调用 expr 指令计算表达式的值，然后使用命令替换将运算结果赋值给相应的变量。其他代码则使用方括号的方式完成表达式的计算和赋值。

- 程序实际完成$S=3(x^y)+4x^2+5y+6$ 的数学表达式运算并输出结果。



4.7　输入和输出

输入指的是 shell 程序读入数据。有从文件读取、从用户输入读取等方式读入数据。输出指的是 shell 程序运行结果的处理，可以显示到屏幕或保存到文件。本节将讲述 shell 程序的输入和输出。

4.7.1　echo 命令输出结果

echo 命令可以输出文本或变量的值，是 shell 中最常用的输出方式。结果可以输出到终端，也可以写入到文件。该命令的用法如下所示。

```
echo $str                    #将结果输出到终端。
echo $str >file              #将结果保存到文件 file 中，如果没有文件会新建一个文件。如果已经有
                             #文件则会覆盖以前的文件。
echo $str >>file             #将结果追加到文件 file 中。
```

echo 输出的内容可以有下面的格式控制字符。

- \c：末尾加上\c 表示这一行输出完毕以后不换行。
- \t：输出一个跳格，相当于按下“Tab”键。
- \n：输出一个换行。

需要注意的是，如果要输出特殊字符，必须加-e 选项，否则输出的结果中会直接输出字符。加-n 选项可以禁止 echo 输出后的换行。例如下面是使用 echo 进行输出的例子。

① 从主菜单中打开一个终端。

② 在终端中输入下面的命令，输出一行文本。

```
echo "hello ,Beijing"
```

③ 按“Enter”键，运行结果如下所示。

```
hello ,Beijing
```

④ 在输出内容中加入一个换行，输入如下所示命令。

```
echo "hello ,\nBeijing"
```

⑤ 按“Enter”键，运行结果如下所示，换行符\n 被直接输出，并没有换行。

```
hello ,\nBeijing
```

⑥ 如果需要在输出内容中显示换行，需要在 echo 后加-e 选项。输入下面的命令。

```
echo  -e "hello ,\nBeijing"
```

⑦ 按“Enter”键，运行结果如下所示。换行符\n 处输出了一个换行。

```
hello ,
Beijing
```

⑧ 在文本中输出几个跳格，输入下面的命令。

```
echo  -e "hello ,\t\t\tBeijing"
```

⑨ 按“Enter”键，运行结果如下所示。

```
hello ,          Beijing
```

⑩ 将结果输出到文件。在终端中输入下面的命令。

```
echo "hello ,Beijing ." >a.txt
```

⑪ 按“Enter”键。在终端中没有输出显示。输入 vim a.txt 命令，用 vim 查看 a.txt 文件。

可以发现文件 a.txt 中有下面的文本。

```
hello ,Beijing .
```

⑫ 再次向这个文本中输出结果，在终端中输入如下的命令。

```
echo "abcde" >a.txt
```

⑬ 按“Enter”键。在终端中没有输出显示。输入 vim a.txt 命令查看文件 a.txt。可以发现文件中有下面的文本。第二次的输出覆盖了第一次输出时建立的文件。

```
abcde
```

⑭ 用追加的方法向这个文本中输出结果，在终端中输入如下的命令。

```
echo "hijkl" >>a.txt
```

⑮ 按“Enter”键。在终端中没有输出显示。输入 vim a.txt 命令查看文件 a.txt。文本的内容如下所示，第二次的输出追加到了第一次输出结果的末尾。

```
abcde
hijkl
```

4.7.2 read 命令读取信息

read 命令可以从键盘或文件中读入信息，并赋给一个变量。read 命令读取信息的各种方法如下所示。

- 如果只读入一个变量，会把键盘上输入的所有信息赋值给这个变量。按“Enter”键结束输入。
- 如果输入多个变量，用“Space”键将输入的变量隔开。如果输入变量的个数多于需要读取变量的个数，将会把剩余的变量赋值给最后一个变量。
- 在读取语句后面添加<filename，表示从文件中读取数据，并且赋值给变量。

例如下面的操作，就是从键盘或文本文件读取变量。

① 从主目录中打开一个终端。

② 在主目录中输入下面的命令，读取一个变量赋值给 A。

```
read A
```

③ 按“Enter”键，终端会等待用户的输入。在终端中输入下面的字符。

```
asdf
```

④ 按“Enter”键，再输入 echo $A 命令输出变量的值。显示的结果如下所示。

```
asdf
```

⑤ 读取多个字符串的变量。在终端中输入 read A 命令。然后按“Enter”键，在光标后面输入下面的字符串。

```
asd fgh jkl
```

⑥ 在终端中输入 echo $A 命令显示这个变量，按“Enter”键后显示的结果如下所示。

```
asd fgh jkl
```

⑦ 读取多个变量。在终端中输入下面的命令，然后按“Enter”键。

```
read A B C
```

⑧ 在终端中等待光标后面输入下面的字符串。

```
aaa sss ddd
```

⑨ 在终端中分别输入下面的命令输出变量的值。然后按“Enter”键。

```
echo $A
echo $B
echo $C
```

⑩ 三次命令的运行结果分别如下所示。

```
aaa
sss
ddd
```

⑪ 如果输入的数据多于需要读取的字符串，将会把多余的输入信息赋值给最后一个变量。在终端中输入下面的命令。

```
read A B
```

⑫ 按“Enter”键执行命令，在光标处输入下面的字符串。

```
aaa bbb ccc ddd
```

⑬ 输出变量。在终端中输入 echo $A 命令，然后按“Enter”键，会输出 aaa。再输入 echo $B 命令，按“Enter”键执行，终端中输出的结果如下所示。

```
bbb ccc ddd
```

⑭ 从文件中读取信息。在终端中输入 vim 命令，打开 vim。

⑮ 在 vim 中按“i”键进入插入模式。然后输入下面的字符串。

```
aaa bbb
```

⑯ 按“Esc”键进入命令模式。输入:w a.txt 命令，再按“Enter”键，保存这个文件。再输入:q 命令，退出 vim。

在终端中输入下面的命令，从文本中读取字符串并赋值给变量。

```
read A B <a.txt
```

输出变量。在终端中输入命令 echo $A 命令，然后按“Enter”键，终端中会显示 aaa。输入 echo $B 命令，然后按“Enter”键，终端中会显示 bbb。

4.7.3　文件重定向

文件重定向指的是，在执行命令时指定命令的输入、输出和错误的输出和输入方式。例如，可以命令的结果输出到一个文件中。表 4.2 列出了文件重定向的常见使用方法。

表 4.2　文件重定向

command > filename	把标准输出重定向到一个文件中
command >> filename	把标准输出以追加的方式重定向到一个文件中
command 1 > fielname	把标准输出重定向到一个文件中
command > filename 2>&1	把标准输出和标准错误一起重定向到一个文件中
command 2 > filename	把标准输出的错误重定向到一个文件中
command 2 >> filename	把标准输出以追加的方式重定向到一个文件中
command >> filename 2>&1	把标准输出和标准错误一起追加到一个文件中
command < filename >filename2	把 command 命令以 filename 文件作为标准输入，以 filename2 文件作为标准输出
command < filename	把 command 命令以 filename 文件作为标准输入

下面是在命令中使用文件重定向的例子。

① 从主菜单中打开一个终端。

② 在终端中输入下面的命令，查看当前的文件夹，把结果保存到文件 a.txt 中。

```
ls > a.txt
```

③ 按“Enter”键以后，命令执行没有显示结果。在终端中输入 vim a.txt 命令，可以发现文件 a.txt 中有上一命令的文件列表。

④ 将上一步骤输出的结果作为命令的输入。在终端中输入下面的命令。

```
read A <a.txt
```

⑤ 按“Enter”键执行命令。这时 read 命令会从文件 a.txt 中读取一个字符串赋值给 A。显示这个变量，输入 echo $A 命令，然后按“Enter”键。这是会显示文件 a.txt 中的第一个字符串，也就是当前用户目录下的第一个文件名，结果如下所示。

```
01.c
```

⑥ 输入和输出都是文件。例如可以用下面的命令，将文件 a.txt 中的所有小写字母转换成大写字母，然后保存到文件 b.txt 中。

```
tr "[a-z]" "[A-Z]" <a.txt >b.txt
```

⑦ 按“Enter”键执行这个命令。然后输入 vim b.txt 命令，查看文件 b.txt 中的内容，可以发现文件中所有的字母都是大写。

⑧ 在 vim 中输入:q 命令，退出 vim。



4.8 引号的使用方法

shell 程序中的单引号、双引号、反引号、反斜线在命令中有特殊含义。本节将讲述这些特殊符号的用法。



注意：反引号指的是“ Tab”键上面的那个符号，在 Linux 系统中常常使用到这个符号。

4.8.1 双引号

双引号表示引用一个字符串。字符串里不能直接使用$、引号、双引号、反斜线、反引号这些特殊符号。如果字符串里没有空格，使用双引号将赋值的字符串引起来，和不使用双引号效果是一样的。当字符串中有空格时，用双引号表示引号中的内容为一个字符串。下面的操作是在字符串中使用双引号。

① 从主菜单中打开一个终端。

② 在终端中输入下面的命令，对变量 A 进行赋值。

```
A="asd fgh jkl"
```

③ 在赋值时，可以用双引号把字符串引起来。输入命令 echo $A，显示变量$A 的结果，终端的结果如下所示。

```
asd fgh jkl
```

④ 在终端中输入下面的命令，对变量 A 进行赋值。字符串中有空格，但是没有引号。

```
A=asd fgh jkl
```

⑤ 按“Enter”键执行这个命令。终端显示命令错误，终端会认为 A=asd 是一个完整的命令，而后面的 fgh jkl 是一个不可识别的命令。

4.8.2　单引号

单引号表示引用一个字符串，用法和双引号是相似的。如果双引号中再使用单引号，则输出时会输出这个单引号。例如输入命令 A="asd'fgh'"对变量 A 赋值。再输入命令 echo $A。命令的显示结果如下所示。

```
asd'fgh'
```



注意：单引号与双引号的区别是：双引号中出现的 shell 变量会被替换为变量的值，而单引号则不会发生变量替换。例如：echo “my home directory is $HOME”**会将 $HOME 替换为当前用户的主目录的路径，而** echo ‘my home directory is $HOME’**打印的结果为：**my home directory is $HOME。

4.8.3　反引号

反引号用于执行引号中的系统命令，然后将命令的结果返回。这个返回的结果可以赋值给一个变量。例如，下面的例子是使用反引号将命令的执行结果赋值给变量。

① 从主菜单中打开一个终端。

② 在终端中输入下面的命令。需要注意的是，命令中的是反引号而不是单引号。

```
echo `date`
```

③ 命令的运行结果如下。

```
12 月 10 15:06:08 CST 2011
```

④ 在终端中输入下面的命令。表示执行 ls 命令，再把结果赋值给变量 A。

```
A=`ls`
```

⑤ 输入 echo $A 命令，输出变量 A 的值，结果如下所示。

```
01.c 02.c 03.c 03.c~ 04.txt 1.txt 4.1.sh 4.2.sh 4.3.sh 4.4.sh
```



提示：反引号经常用于将一个命令的输出结果当做另一个命令的参数。

4.8.4　反斜线

反斜线用于对特殊字符进行转义。如果字符串中含有&、*、+、^、$、`、"、|、?这些特殊符串，shell 会认为这些符串代表着相应的运算。可以使用反斜线对这些符串进行转义。例如下面的例子是在字符串中用反斜线处理特殊符号。

① 从主菜单中打开一个终端。

② 定义含有三个引号的字符串。引号是一个特殊字符，如果直接对一个变量赋值 3 个引号，命令会发生错误。需要用反斜线对输入的引号进行转义。输入的命令如下所示。

```
A=\"\"\"
```

③ 输入命令 echo $A，再按“Enter”键执行，会输出三个引号，结果如下所示。

```
"""
```

④ 输出一个字符串$$。在命令中输入 echo $$，终端会显示当前进程的 ID。需要使用\进行转义，输入命令 echo \$$。

4.9 测试语句

这里所说的测试是指对变量的大小、字符串、文件属性等内容进行判断。test 命令可以用于字符串、数字、文件状态等内容的测试。本节将讲述 shell 的测试语句。

提示：test 命令经常用于在 shell 脚本中与 if 语句连用进行条件判断。

4.9.1 文件状态测试

文件状态测试指的是对文件的权限、有无、属性、类型等内容进行判断。与其他语言不同的是，test 命令的测试结果，返回 0 时表示测试成功。返回 1 时表示测试失败。表 4.3 是文件测试的参数列表。

表 4.3 文件状态测试表

-d 测试文件是否是目录文件	-s 测试文件是否非空
-f 测试文件是否是正规文件	-w 测试是否可写
-L 测试文件是否是符号链接	-u 测试文件是否有 suid 位设置
-x 测试文件是否可执行	-r 测试文件是否可读

例如下面的例子使用这些测试参数，对文件的属性进行测试。

① 从主菜单中打开一个终端。

② 测试文件/windows 是否是一个目录。在终端中输入下面的命令，然后按“Enter”键。

```
test -d /windows
```

③ 输出测试结果。$?用于保存上一个命令的结果，可以用下面的命令进行输出。

```
echo $?
```

④ 终端中输出结果 1，表明/windows 不是一个目录。

⑤ 测试当前目录下的文件 a.txt 是否可以执行。从表 4.3 中可知，测试是否可以执行的参数是-x，所以输入的命令如下所示。

```
test -x a.txt
```

⑥ 按“Enter”键执行。输入 echo $?命令输出结果。

⑦ 终端中显示的结果是 1。表明文件 a.txt 是不能执行的。

提示：特殊变量$?表示上一个命令的返回值，在 shell 脚本中，进程使用$?判断上一个命令执行是否成功。如果返回值为 0，则表明上一个命令执行成功，否则执行失败。

4.9.2 数值测试

数值测试指的是比较两个数值的大小或相等关系，相当于 C 语言中的比较运算符。shell 程序中的数值测试有下面两种形式。

- 用 test 命令。test 命令和相应的参数可以对两个数值的关系进行测试，使用方法如下所示。

```
test  第一个操作数  数值比较符 第二个操作数
```

- 用中括号代替 test 命令。这种方法和 test 命令的原理相同，使用方法如下所示。

```
[ 第一个操作数　数值比较符 第二个操作数　]
```



注意：左方括号[后面和右方括号]前面分别有一个空格。

数值比较符相当于 C 中的数据比较符号，不同的是需要使用字符串写出。数值比较符如表 4.4 所示。

表 4.4　数值比较符

-eq：两个数是否相等	-ne：两个数是否不相等
-le：第一个数是否不大于第二个数	-ge：第一个数是否不小于第二个数
-gt：第一个数是否大于第二个数	-lt：第一个数是否小于第二个数

例如下面的例子是使用数值比较符进行数字测试。

① 从主菜单中打开一个终端。

② 测试 3 和 5 是否相等。在终端中输入下面的命令。

```
test 3 -eq 5
```

③ 按“Enter”键运行这个命令。然后输入$?命令输出测试结果。终端中显示的结果为 1，表明测试结果是 3 和 5 不相等。

④ 测试 10 是否小于 12。从表 4.4 中可知，小于测试的参数是-lt。在终端中输入下面的命令。

```
test 10 -lt 12
```

⑤ 按“Enter”键运行这个命令。然后输入$?命令输出上一次测试的结果。终端中显示的结果为 0，表明 10 小于 12 为真。

⑥ 上面的测试也可以写成下面的命令。需要注意的是，[后面一定要有个空格。

```
[ 10 -lt 12 ]
```

4.9.3　字符串测试

所谓字符串测试，指的是比较两个字符串是否相等，或者判断一个字符串是否为空。这种判断常用来测试用户输入是否符合程序的要求。字符串测试有下面 4 种常用的方法。

- test 字符串比较符　字符串
- test　字符串 1　字符串比较符　字符串 2
- [字符串比较符　字符串　]
- [字符串 1　字符串比较符　字符串 2]

字符串比较符有如下所示的 4 种。

- =：测试两个字符串是否相等；
- !=：测试两个字符串是否不相等；
- -z：测试字符串是否是空字符串；
- -n：测试字符串是否非空字符串。

下面的例子是进行字符串测试。

① 从主菜单中打开一个终端。

② 用 read 命令读入两个变量，在终端中输入下面的命令。

```
read A B
```

③ 按“Enter”键运行命令，在终端中输入下面的字符串。

```
aaa bbb
```

④ 测试变量 A 与变量 B 是否相等。在终端中输入下面的命令。

```
test $A = $B
```

⑤ 按“Enter”键运行这个命令。然后输入 echo $?命令输出结果。显示的结果为 1，表明两个字符串不相等。

4.9.4 逻辑测试

逻辑测试指的是将多个条件进行逻辑运算，常用做循环语句或判断语句的条件。shell 程序中有下面三种逻辑测试。

- -a：逻辑与，操作符两边均为真时结果为真，否则为结果为假。
- -o：逻辑或，操作符两边至少一个为真时，结果为真，否则为假。
- ！：逻辑否，只有条件为假时，返回结果为真。

例如下面的例子，可以判断主目录中的文件 a.txt，是否同时可写并且可执行。

① 从主菜单中打开一个终端。

② 在终端中输入下面的命令，用-a 表示需要同时满足两个测试条件。

```
[ -w a.txt -a -x a.txt ]
```

③ 按“Enter”键执行这个命令。然后输入 echo $?命令输出测试结果，终端中显示 1。表明了测试结果为假，因为文件 a.txt 是不可写的。

4.10 流程控制结构

所谓流程控制，指的是使用逻辑判断，针对判断的结果执行不同语句或不同的程序部分。这种结构是所有编程语言的重要组成部分。本节将讲述判断、循环等流程控制结构。

注意：shell 中的流程控制结构，实际上是使用 shell 的内部命令来实现的。使用方法与其他编程语言中的流程控制语句类似。

4.10.1 if 语句

if 语句是最常用的条件判断语句，是通过一个条件的真假来决定后面的语句是不是执行。最简单的 if 条件语句如下所示。

```
if 条件
  then 命令 1
fi
```

在这种结构中，先执行条件判断，如果条件结果为真，则执行 then 后面的语句，一直到 fi。如果条件为假，则跳过后面的语句，执行 fi 后面的语句。

如果条件判断的结果，只可能是真或假两种值，可以使用下面的结构。

```
if 条件
then 命令 1
else 命令 2
fi
```

在这种结构中，先对条件进行判断，如果判断结果为真，则执行 then 后面的语句。如果判断结果为假，则执行 else 后面的语句。如果判断的结果有多种可能，则使用下面的 if 语句嵌套结构。需要注意的是，if 结构必须要有 fi 进行结束。

```
if 条件 1
then 命令 1
elif 条件 2
then 命令 2
else 命令 3
fi
```

if 语句也可以将 then 写在 if 条件之后，中间用分号隔开。这种语句的形式如下所示。

```
if 条件 1;then 命令 1
elif 条件 2 ;then 命令 2
else 命令 3
fi
```



提示：if 语句的结尾时 fi，即 if 的反序。

范例 4-4 所示的例子，是编写一个 shell 程序。程序从参数中读取一个数字，然后判断这个数字是奇数还是偶数。

【范例 4-4】shell 脚本中使用 if 语句进行条件判断，实例代码如下所示。

实例代码 4-4

```
01  #!/bin/bash
02  #4.4.sh
03  i=$[  $1 % 2 ]
04  if  test  $i -eq 0 ; then
05    echo Even Number
06  else
07    echo Odd Number
08  fi
```

【运行结果】在终端中输入下面的命令，对这个文件添加可执行权限：

```
chmod +x 4.4.sh
```

在终端中输入如下所示的命令运行这个程序，在命令中加入一个参数。

```
./4.4.sh 4
```

程序的执行结果如下所示。

```
Even Number
```

【代码解析】

- 代码第 03 行完成对命令行中输入的参数与 2 取模运算。
- 代码第 04~08 行判断变量$i 的值，并打印第一个命令行参数是奇数还是偶数。

4.10.2　for 语句

for 语句是一种常用的循环语句，实现在一个值列表中的循环功能。下面是 for 语句的使用方法。

```
for 变量名 i  n 列表
do
命令 1
命令 2…
```

```
done
```

范例 4-5 所示是一个简单的 for 循环程序，作用是用循环的方法输出列表中的数值。

【范例 4-5】使用 for 循环语句输出字符列表中的字符，实例代码如下所示。

实例代码 4-5

```
01  #!/bin/bash
02  #4.7.sh for
03  for char in a s d f g                    #开始 for 循环。
04  do                                       #循环体。
05     echo $char
06  done                                     #结束 for 循环，
```

【运行结果】在终端中输入下面的命令，运行这个程序。

```
./4.7.sh
```

程序的运行结果如下所示。

```
a
s
d
f
g
```

【代码解析】

- 代码第 03 行中的“a s d f”表示在进行循环时变量 i 依次取值。
- 被“do”和当“done”包围的命令是 for 语句的循环体。



注意：当 for 语句省略后边的 in 关键字时，将接受输入命令时的参数作为循环变量集。

范例 4-6 所示的一个 for 循环程序，可以输出程序中所有的参数。

【范例 4-6】使用 for 循环语句输出所有的命令行参数，实例代码如下所示。

实例代码 4-6

```
01  #!/bin/bash
02  #4.8.sh for
03  for  str                                 #开始 for 循环。
04  do                                       #循环体。
05     echo $str
06  done                                     #结束 for 循环。
```

【运行结果】在终端中输入下面的命令运行这个程序。

```
./4.8.sh a s d f
```

程序会依次列出所输入的参数，运行结果如下所示。

```
a
s
d
f
```

【代码解析】代码第 03 行代码省略了 in，则 for 循环的变量值来自命令行的输入。

4.10.3 until 语句

until 循环用于执行一个循环体，直至条件为真时停止。这种语言结构如下所示。

```
until 条件
do
命令 1
…
done
```

范例 4-7 所示的实例是用一个 until 循环求出 1 至 100 之间所有整数的和，然后输出结果。在程序中，需要有一个变量用来存放求和的结果，一个变量用于循环的计数。

【范例 4-7】使用 until 循环求 1 到 100 相加的和，实例代码如下所示。

实例代码 4-7

```
01  #!/bin/bash
02  #4.11.sh
03  sum=0
04  i=1
05  until [ $i -gt 100  ]
06  do
07     sum=$[$sum+$i]
08     i=$[$i+1]
09  done
10  echo $sum
```

【运行结果】对这个文件添加可执行权限，在终端中输入下面的命令。

```
chmod +x 4.11.sh
```

输入下面的命令运行这个程序。

```
./4.11.sh
```

程序求出了 1 到 100 之间的整数和，结果如下所示。

```
5050
```

【代码解析】

- 代码第 03 行和代码第 04 行分别定义了两个 shell 变量，“sum”用于保存计算结果，而变量“i”则为循环标志变量。
- 代码第 07 行和代码第 08 行分别实现求和与循环标志变量的自动操作。



4.11　综合练习

shell 程序以简单的代码调用各种系统命令实现强大的功能，在系统设置、服务管理方面有着重要的应用。本节讲述一个完整的 shell 程序实例，自动完成计算机上 U 盘的挂载与文件复制的功能。

4.11.1　程序的功能

在 Linux 系统中，如果计算机上插入一个 USB 设置，需要用挂载命令才能实现这个设备的加载。可以把 USB 的挂载与文件的复制写成一个 shell 程序，这样可以通过程序的运算自动完成很多步骤的操作，这个程序的功能如下所示。

（1）运算程序时，提示用户输入“y”或“n”，确定是不是挂载 USB 设备。

（2）如果用户输入“y”，则挂载这个 USB 设备。

（3）提示用户输入“y”或“n”，确定是不是复制文本。

（4）如果用户输入“y”，则显示文件的列表，然后提示用户是否复制文件。

（5）程序根据用户输入的文件名复制相应的文件，然后提示是否将计算机中的文件复制到

USB 中。

（6）完成文件的复制以后，提示用户是否卸载 USB 设备。

4.11.2 编写程序的代码

根据 4.10.1 节的功能分析，编写出范例 4-8 所示的程序的代码。在编程时，需要注意判断语句、循环语句的使用，对用户的输入做出正确的判断。

【范例 4-8】使用 shell 脚本实现 USB 设备加载与文件复制，实例代码如下所示。

实例代码 4-8

```
#!/bin/bash
#autousb

echo  "welcome to use AUTOUSB"
echo  "do you want load usb(y/n)?"
read  ANSWER

if  [ $ANSWER = "Y" -o $ANSWER = "y" ]
  then mount -t vfat /dev/sda1 /mnt/usb
  echo  "do you want copy files to  /root(y/n)?"
  read  ANSWER
  while [  $ANSWER  =  "y"  -o  $ANSWER =  "Y" ]
  do
    ls -a /mnt/usb
    echo "type the filename you want to copy"
    read  FILE
    cp /mnt/usb/"$FILE"  /root
      if [  $? -qe 0 ];then
        echob  " copy finished"
      else
        echob  " copy errored "
      fi
    echo  "any other files(y/n)?"
    read  ANSWER
  done
fi

echo   "do you want to copy files to usb(y/n)?"
read  ANSWER
while  [  $ANSWER  =  "y"  -o  $ANSWER  =  "Y" ]
do
  ls -a  /root
  echo  "type the filename you want to copy to usb"
  read  FILE
  cp "/root/$FILE"  /mnt/usb
   if [  $? -qe 0 ];then
       echob  " copy file finished"
      else
       echob  " copy file errored "
    fi
  echo  "any other files(y/n)?"
  read  ANSWER
done

echo  "do you want to umount usb(y/n)?"
read  ANSWER
if  [ $ANSWER  =  " y"  -o   $ANSWER  =  " Y" ] ;then
  umount /mnt/usb
  else
```

```
50   echo "Haven't umount!"
51   fi
52   echo "GoodBye!"
```

【运行结果】对这个文件添加可执行权限，在终端中输入下面的命令。

```
chmod +x 4.12.sh
```

输入下面的命令运行这个程序。

```
./4.12.sh
```



4.12 小结

本节讲述了 shell 程序的基本语法。通过本章的学习，可以编写出简单的 shell 程序完成各种系统功能。在 shell 程序的学习中，需要注意局部变量和环境变量和使用、各种符号的使用方法。测试语句与流程控制语句是本章的难点，需要对程序的各种变量进行逻辑判断，执行不同的程序。



4.12 习题

1．编写 shell 脚本程序实现求 1~100 相加的和。

2．编写 shell 脚本程序，提示用户输入两个数字，在程序中比较这两个数字的大小，并且输出较大者。



4.14 面试题

【题目 1】请问在 Bash shell 环境下，当一个命令正在执行的时候，如果按下“Ctrl+Z”组合键，会产生什么结果？

【分析】此题考查 Linux shell 的基础知识。在 Bash shell 中定义了几种常见的组合键，包括 Ctrl+Z、Ctrl+C，以及 Ctrl+\，它们分别有不同的作用。

【题目 2】在 Linux 系统中，用户编写的脚本文件一般以什么开头？

【分析】Linux 系统的 shell 脚本多种多样，需要熟记常见的几种脚本开头格式，例如 Python 脚本、Bash 脚本等。

【题目 3】Linux 系统中，一般把命令 ls 定义为 ls --color 的别名，以便以不同颜色来标识不同类型的文件。但是，如何能够使用原先的 ls 命令？

【分析】Bash shell 是常见的命令行环境，其中提供了丰富的功能，包括命令别名功能。请参考 Bash Shell 的帮助文件，掌握常见的功能。

【题目 4】设计一个 shell 程序，在每月第一天备份并压缩/etc 目录的所有内容，存放在/root/bak 目录里，且文件名为如下形式：yymmdd_etc，yy 为年，mm 为月，dd 为日。shell 程序 fileback 存放在/usr/bin 目录下。

【分析】此题考查如何使用时间作为文件名建立备份文件。因为时间是动态的值，需要使用 Linux 的命令获取时间值，然后利用 shell 脚本的功能，将获取的时间作为文件名建立文件。

第四篇　指针与函数篇

第 8 章　数组与指针

数组在进行软件开发时的使用频率很高，指针则是 C 语言的精华所在，同时也是比较难以掌握的知识点。本章首先分别介绍数组和指针的基本概念和使用方法，然后介绍数组与指针之间的关系，最后通过实例介绍在实际开发中如何灵活地使用数组与指针。本章的主要内容如下：

- 数组与指针的概念与基本操作；
- 数组与指针的关系；
- 二维数组与多维数组；
- 指针与数组的常见问题。



8.1　数组的理解与操作

数组实际上就是一组相同数据类型的变量。如果一个数组中的一个变量也是一个数组，就构成了二维数组。同样的方法，可以构成多维数组。在使用数组以前需要建立一个数组。

8.1.1　什么是数组

可以用一个实例来理解数组。假设在一个程序中要存储 100 个人的年龄，可以写成下面的代码来定义 100 个整型变量。

```
int age1=12;
int age2=18;
int age3=15;
……
int age100=21;
```

如果这样编写程序，程序中会产生大量相似的语句，在调用变量时，变量名没有统一的规则。可以将所有同类的数据存放在一个数组变量中。所有人的年龄可以建立一个数组，然后用编号来表示数组的一个变量。可以用下面的代码来表示这些数据。

```
int a[100];
age[0]=12;
age[1]=18;
age[2]=15;
……
age[99]=21;
```

这样就可以用一个变量名和一个下标来表示数组中的一个变量。例如 age[0]表示数组中的第一个变量，age[1]表示数组中的第二个变量。age 是这个数组的变量名，后面方括号内的数字称为数组下标。例如范例 8-1 所示的程序，定义了一个有 10 个整型变量的数组，然后用循环语句对数组赋值 10~19。再用循环语句输出这些数据变量。

【范例 8-1】使用循环语句为一个包含 10 个元素的数组赋值，实例代码如下所示。

实例代码 8-1

```
01  #include <stdio.h>
```

```
02  int main()
03  {
04    int a[10];
05    int i;
06    for(i=0;i<=9;i++)
07    {
08      a[i]=i+10;
09    }
10    for(i=0;i<=9;i++)
11    {
12      printf("%d  ",a[i]);
13    }
14    return 0;
15  }
```

【执行结果】编译并运行这个程序，结果如下所示。

```
10  11  12  13  14  15  16  17  18  19
```

【代码解析】实例代码 8-1 的代码解析如下：

- 代码第 04 行定义了一个具有 10 个元素的数组 a。
- 代码第 06~09 行使用 for 循环为数组元素赋值。
- 代码第 10~13 行输出数组 a 中所有元素的值。

在使用数组时，需要注意下面这些下标和数据类型的问题。

- 在定义数组时，已经定义了数组的数据类型，在访问时需要按照数组的数据类型进行访问和赋值。
- 对于一个已经定义的数组，数组中所有的元素都是相同的数据类型。
- 数组的下标是从 0 开始的向后排列整数，不能是其他的下标。第一个下标是 0 而不是 1。下标 *n* 表示第 *n*+1 个变量。
- 定义数组 a[n]以后，有 *n* 个元素，但是没有 a[n]这个元素。访问不存在的数组变量时，程序就会发生数组下标越界错误。



注意：数组下标越界是指访问了不存在的数组元素，导致访问了非法的内存区域而产生的错误。在 C 语言中一定要避免内存越界之类的错误。

在本质上，数组是内存上一组同类信息列出的一个表。假设上面例子中的数组 a[10]是从内存地址为 0x1000 的存储单元开始存储的，数组的内容和值可用表 8.1 表示（假设使用 32 位 CPU 的 Linux 操作系统）。

表 8.1　数组的存储与值

变量	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
值	10	11	12	13	14	15	16	17	18	19
存储地址	0x1000	0x1004	0x1008	0x100C	0x1010	0x1014	0x1018	0x101C	0x1020	0x1024

8.1.2　数组的定义与访问

数组的定义指的是在内存中开辟一块存储空间，生成一个空数组。数组的定义与变量的定义相似，需要指定数据类型的数组元素的多少，数组元素数目用中括号括起来。例如下面的代码定义不同类型的数组。

```
01  int a[10];                  /*定义 10 个元素的整型数组。*/
02  float f[20];                /*定义有 20 个元素的浮点型数组。*/
```

```
03   char s[5];                    /*定义有 5 个元素的字符型数组。*/
```



提示：数组在定义时，可以不指定数组元素的个数，在访问时可以动态改变数组中元素的个数。例如下面的代码定义一个元素个数不定的字符型数组。

```
01 char a[];                /*定义一个含有不定个数元素的字符数组。*/
```

访问数组中的元素很简单。用数组名称与下标就可以访问指定的数组元素。例如下面的代码输出数组元素的值和对数组元素进行赋值。

```
01   printf("%d",a[2]);     /*输出数组 a 中第 3 个元素的值。*/
02   a[3]=5;                /*对数组中的第 4 个元素进行赋值。*/
```

8.1.3　数组使用实例

数组可以实现数组元素的循环访问功能，和循环语句一起用可以方便地实现数组元素的写入和读取。范例 8-2 所示的实例，定义一个字符型数组，然后从键盘上输入字符存储到这个数组中。当输入一个“#”时停止输入，然后用循环语句输出这些字符。

【范例 8-2】将输入信息保存到数组中，遇到“#”停止输入，实例代码如下所示。

实例代码 8-2

```
01  #include <stdio.h>
02  int main()
03  {
04    char a[100] = {0};                  /*定义一个字符型数组并且赋初值。*/
05    char c='a';                         /*定义一个字符变量，赋一个初值。*/
06    int i=0 ;                           /*定义一个计数变量，实现下标的计数。*/
07    while(1)                            /*进入一个循环。*/
08    {
09      scanf("%c",&c);                   /*输入一个字符。*/
10      if(c=='#')                        /*如果字符为#则中断循环。*/
11      {
12        break;                          /*结束循环。*/
13      }
14      a[i]=c;                           /*把输入的变量赋值给数组中的一个元素。*/
15      i++ ;                             /*计数变量自加。*/
16    }
17    i=0;                                /*输出时，计数变量赋值为 0。*/
18    while(a[i]!=0)
19    {
20      printf("%c",a[i]);                /*输出数组的中字符。*/
21      i++ ;                             /*下标自加。*/
22    }
23    return 0;
24  }
```

【执行结果】输入下面的命令，编译这个程序。

```
gcc 8.2.c
```

输入下面的命令，运行这个程序。

```
./a.out
1
b
3
#
1
b
```

```
3
```

在程序中输入一个字符，然后按“Enter”键。输入多个字符以后，输入“#”结束输入。程序会输出这些字符。

【代码解析】这个程序是用井号“#”来结束循环输入的。每次输入一个字符，判断这个字符是不是“#”。如果是“#”则用 break 语句来中断循环。在输出时，判断输出的字符是不是存在。在定义数组时，为数组中的元素赋初值 0。



注意：由于数组定义后无法修改数组的大小，所以，在定义数组时要充分考虑数组能够存储的元素个数。



8.2 指针

指针是一种特殊的数据类型，用来存储一个变量的内存地址。通过一个指针，可以访问这个指针所指向的变量。在使用指针时，需要考虑到变量的存储关系。

8.2.1 指针的理解

内存中的数据通常是按照字节的方式存放在计算机的内存中的，每个字节都对应了一个内存中的编号，称为内存地址。表 8.2 展示了内存地址和变量的内容之间的关系。

表 8.2 指针和变量

内存地址	内存中的变量
1005	1
1006	2
1007	3
1008	b
1009	c
1010	d

在前面的章节讲到的变量都是使用一个名字表示的，没有涉及变量在内存中占用的地址。在前面讲到的例子中，一个变量存放的数据可以通过变量名称直接得到。指针是一种特殊的变量，它存放的数据不是直接的数据，而是内存地址。换句话说，通过指针变量可以访问一个内存地址或其他变量。



提示：指针即内存地址，通过指针可以直接操作内存中的数据。

8.2.2 指针操作符

指针操作符有*与&两个，分别实现取变量和取地址的操作。程序中就是通过这两个操作符实现指针的定义与访问的。“&”可以实现取一个变量的地址的功能。取出的变量地址可能对应一个很复杂的数据类型，但是操作时并不关心数值的多少，只需要保存到一个指针变量。例如下面的代码定义了一个指针变量并且取出另一个变量的地址给指针变量赋值。

```
01   int *p;                /*定义一个指针变量。*/
02   int i=5;               /*定义一个整变量。*/
03   p=&i;                  /*取变量 i 的地址赋值给 p。*/
```

“*”实现取一个指针所指向的变量的功能。例如下面的代码，就是通过一个变量的指针来访问变量。

```
01   int *p;                /*定义一个指针变量。*/
02   int i=5 , j;           /*定义两个整变量。*/
03   p=&i;                  /*取变量 i 的地址赋值给 p。*/
```

```
04   j=*p;                    /*取指针 p 所指向变量的值赋值给 j。*/
```



注意：在定义指针变量时，*表示定义的变量为指针变量，而类似 j=*p 的操作中，*则表示取地址空间的变量值。

8.2.3　指针的定义与访问

指针也有数据类型。指针的数据类型需要和它所指向的变量类型匹配。定义指针的方法是用定义变量的方法，在变量名称前面加一个“*”。例如下面的代码是指向不同数据类型的指针。

```
01   int *p1;                 /*定义一个指向整型变量的指针。*/
02   char *p2;                /*定义指向一个字符型变量的指针。*/
03   float *p3;               /*定义一个指向浮点型变量的指针。*/
```

一个变量是用“&”取地址的。一个指针变量可以用“*”来取这个变量指向的数值。例如范例 8-3 所示的代码是使用指针访问变量的实例。

【范例 8-3】指针的使用案例，实例代码如下所示。

实例代码 8-3

```
01   #include <stdio.h>
02   int main()
03   {
04     int i ,j;                          /*定义两个整型变量。*/
01     int *p;                            /*定义一个指向整型变量的指针。*/
02     char a ,b;                         /*定义两个字符变量。*/
03     char *q;                           /*定义一个指向字符变量的指针。*/
04     i=5;                               /*i 赋值为 5。*/
05     p=&i;                              /*取 i 的地址赋值给 p。*/
06     j=*p;                              /*取指针 p 指向的变量值赋值给 j。*/
07     a='w';                             /*a 赋值为 w。*/
08     q = &a;                            /*取 a 的地址赋值给指针变量 q。*/
09     b=*q;                              /*取指针 q 指向的变量值赋值给 b。*/
10     printf("%d  %d  %d \n",i ,j ,*p);  /*输出结果。*/
11     printf("%c  %c  %c \n",a ,b ,*q);  /*输出结果。*/
12     return 0;
13   }
```

【执行结果】输入下面的命令，编译这个程序。

```
gcc 8.3.c
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示。

```
5  5  5
w  w  w
```

从结果可知，通过一个指向变量的指针，可以访问被指向变量的值。在 C 语言的函数参数中，常常通过指针来访问一个变量的值。

【代码解析】实例代码 8-3 的代码解析如下：

- 代码第 01 行和第 03 行分别定义了两个指针变量。
- 代码第 05 行和第 08 行将变量的地址赋值给指针变量。

- 代码第 10 行和第 11 行输出指针变量和非指针变量的值。

8.2.4　指针使用实例

在程序中可以使用指针来访问变量的值。本节将讲述一个指针使用实例，通过指针所指向的变量比较两个整数的大小，然后从小到大输出两个整数。

【范例 8-4】通过指针所指向的变量比较整数的大小，实例代码如下所示。

实例代码 8-4

```
#include <stdio.h>
int main()
{
   int i ,j;                                  /*定义两个整型变量。*/
   int *p,*q,*temp;                           /*定义指向两个整型变量的指针变量。*/
   printf("please input the first number: \n");      /*提示输入。*/
   scanf("%d",&i);                                   /*输入一个数值。*/
   printf("please input the second number: \n") ;
   scanf("%d",&j);
   p=&i;                                      /*取 i 的地址赋值给指针 p。*/
   q=&j;                                      /*取 j 的地址赋值给指针 q。*/
   if(*p>*q)                                  /*判断这两个指针指向值的大小。*/
   {                                          /*用一个中间指针变量交换两个指针。*/
     temp=p;
     p=q;
     q=temp;
   }
   printf("%d  %d\n",*p,*q);                  /*输出结果。*/
   return 0;
}
```

【执行结果】输入下面的命令，编译这个程序。

```
gcc 8.4.c
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序运行时，输出提示，这时输入一个变量 5，然后按“Enter”键。然后再输入一个变量 3，然后按“Enter”键。这时程序显示的结果如下所示。

```
3  5
```

【代码解析】本程序是通过指针所指向的值来实现两个整数的排列的。通过比较指针所指向变量的大小，用一个中间指针变量交换两个指针变量的值。



8.3　数组与指针

数组与指针的联系非常紧密。除了可以用下标访问数组元素以外，也可以用指向数组的指针访问数组中的元素。本节将讲解数组与指针的关系。

8.3.1　数组与指针的关系

定义一个数组的时候，定义这个数组的头指针，然后分配若干个存储单元。定义的数组名称是可以直接赋值给一个指针的，而这个指针可以指向这个数组的第一个元素。范例 8-5 所示的程序可以说明数组与指针的这种关系。

【范例 8-5】数组与指针的关系实例代码如下所示。

实例代码 8-5

```
01  #include <stdio.h>
02  int main()
03  {
04     int *p;                          /*定义一个指向整型变量的指针。*/
05     int a[3];                        /*定义一个整型数组。*/
06     a[0]=10;                         /*对数组的变量赋值。*/
07     a[1]=11;
08     a[2]=12;
09     p=a;                             /*将数组赋值给一个指针。*/
10     printf("%d\n",*p);               /*输出这个指针指向的内容。*/
11     p++;                             /*指针向后移动一个单元。*/
12     printf("%d\n",*p);               /*输出指针指向的内容。*/
13     p++;                             /*指针向后移动一个单元。*/
14     printf("%d\n",*p);               /*输出指针指向的内容。*/
15     return 0;
16  }
```

【执行结果】编译并运行这个程序，结果如下所示。

```
10
11
12
```

从程序的结果可知，数组名称相当于指向一组连续数据指针。用指针来访问数组的元素与下标访问量等效。

【代码解析】

- 代码第 09 行将数组的首地址赋值给指针变量 p。
- 代码第 10 行、第 12 行和第 14 行分别输出数组的第 0 个元素的值、第 1 个元素的值和第 2 个元素的值。



提示：数组名即为数组的头指针。

8.3.2 指针的算术运算

指针可以做加法和减法的算术运算，相当于指针的位置从这一位置向后或向前移动若干个单元。这里移动的单元，指的是可以存储相应的变量的内存空间。下面的实例可以说明指针算术运算的作用。



提示：指针的加减法运算是基于指针所代表的存储单元为单位，而不是以字节为单位。

【范例 8-6】指针的算术运算，实例代码如下所示。

实例代码 8-6

```
01  #include <stdio.h>
02  int main()
03  {
04     int i;                           /*定义一个循环变量 i。*/
05     int *p;                          /*定义一个指向整型变量的指针。*/
```

```
06     int a[10];                              /*定义一个整型的数组。*/
07     for(i=0;i<10;i++)                       /*变量 i 从 0~9 执行循环。*/
08     {
09       a[i]= i +10;                          /*对数组进行赋值。*/
10     }
11     p=a;                                    /*将数组 a 的指针赋值给 p。*/
12     for(i=0;i<10;i++)                       /*进行 10 次循环。*/
13     {
14       printf("%d  ",*p);                    /*输出指针 p 指向的值。*/
15       p++;                                  /*指针 p 向后移动一个单元。*/
16     }
17      printf("\n",*p);                       /*输出一个换行。*/
18      p=p-4;                                 /*指针向前移动 4 个单元。*/
19      printf("%d  ",*p);                     /*输出指针 p 指向的值。*/
20      p=p-3;                                 /*指针向前移动 3 个单元。*/
21      printf("%d  ",*p);                     /*输出指针 p 指向的值。*/
22      p=p+5;                                 /*指针向后移动 5 个单元。*/
23      printf("%d  \n",*p);                   /*输出指针 p 指向的值。*/
24      return 0;
25   }
```

【执行结果】输入下面的命令，编译这个程序。

```
gcc 8.6.c
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示。

```
10  11  12  13  14  15  16  17  18  19
16  13  18
```

【代码解析】代码第 18 行 第 20 行和第 22 行通过指针的算术运算来访问对应的数组元素。这种访问数组元素的方式与使用下标的方式效果相同。

8.3.3　字符数组与字符串

字符数组是一种特殊的数组。定义一个字符数组以后，这个字符数组会返回一个头指针。可以根据这个头指针来访问数组中的每一个字符。而 scanf()输入函数和 printf()输出函数可以输入或输出一个字符串。范例 8-7 所示的实例是字符数组的使用方法。

【范例 8-7】字符数组与指针的关系，实例代码如下所示。

实例代码 8-7

```
01   #include <stdio.h>
02   int main()
03   {
04     char a[30];                                  /*定义一个字符数组。*/
05     char *p;                                     /*定义一个指向字符变量的指针。*/
06     int i;                                       /*定义一个计数器变量。*/
07     printf("please input a string:\n") ;         /*提示输入。*/
08     scanf("%s",a) ;                              /*输入一个字符数组。*/
09     printf("result:\n") ;
10     printf("%s\n",a) ;          /*输出一个字符数组。*/
11     i=0;                        /*循环计数器赋初值。*/
12     while(a[i]!=NULL)           /*判断相对应的字符是不是为空。*/
13     {
```

```
14        printf("%c",a[i]) ;          /*输出这个字符。*/
15        i++;                         /*计数器自加。*/
16      }
17      printf("\n") ;                 /*输出一个换行。*/
18      p=a;                           /*数组的头指针赋值给指针 p。*/
19      printf("%s\n",p) ;             /*输出变量 p 的内容。*/
20      while(*p!=NULL)                /*判断 p 指向的内容是不是为 NULL，进行 while 循环。*/
21      {
22        printf("%c",*p) ;            /*输出指针 p 指向的一个字符。*/
23        p++;                         /*指针 p 指向下一个变量。*/
24      }
25      return 0;
26    }
```

【执行结果】输入下面的命令，编译这个程序。

```
gcc 8.7.c
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序输出下面的提示，要求输入一个字符串。

```
please input a string:
```

这时输入下面的字符串，然后按“Enter”键。

```
asdfgh
```

程序的运行结果如下所示。

```
please input a string:
asdfgh
result:
asdfgh
asdfgh
asdfgh
```

【代码解析】上面的代码使用三种情况演示了如何访问数组中的元素。代码第 12~16 行使用数组下标访问数组，代码第 19 行使用字符串的方式打印字符数组中的所有元素，代码第 20~24 行使用指针访问数组元素。



注意：C 语言中没有字符串数据类型，可以使用字符数组存储和操作字符串。



8.4 二维数组与多维数组

如果数组的每一个元素是一个一维数组，则这个数组就是一个二维数组。在二维数组中，变量可以理解成行和列的关系。用同样的方法，可以构造出多维数组。

8.4.1 二维数组的理解

二维数组可以理解成一个多行多列的表格，每一个单元格中存储了一个变量。可以同一维数组一样定义二维数组。下面的代码定义了一个 4 行 5 列的整型数组。

```
01  int a[4][5];
```

可用表 8.3 来表示这个数组的行和列存储关系。

如表 8.3 所示，可以用数组的下标访问二维数组的每一个元素。范例 8-8 所示的代码使用



for 循环对这个二维数组进行赋值与输出。

提示：二维数组可以认为是数组的数组，即一维数组的元素又是一个一维数组。

表 8.3　二维数组

列 / 行	0	1	2	3	4
0	a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
1	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
2	a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]
3	a[3][0]	a[3][1]	a[3][2]	a[3][3]	a[3][4]

【范例 8-8】使用 for 语句为二维数组赋值并输出，实例代码如下所示。

实例代码 8-8

```
01  #include <stdio.h>
02  int main()
03  {
04     int i=10 , m , n ;                          /*定义程序中的变量。*/
05     int a[4][5];                                /*定义一个四行五列的二维数组。*/
06     for(m=0;m<4;m++)                            /*用变量 m 从 0 到 3 进行行循环。*/
07     {
08        for(n=0;n<5;n++)                         /*变量 n 从 0 到 4 进行列循环。*/
09        {
10           a[m][n]=i;                           /*数组中变量赋值。*/
11           i++;                                 /*i 自加。*/
12        }
13     }
14     for(m=0;m<4;m++)                            /*变量 m 的行循环。*/
15     {
16        for(n=0;n<5;n++)                         /*变量 n 的列循环。*/
17        {
18          printf("a[%d][%d]=%d  ",m,n,a[m][n]); /*输出变量名变量值。*/
19        }
20        printf("\n");                            /*每行输出一个换行。*/
21     }
22     return 0;
23  }
```

【执行结果】输入下面的命令，编译这个程序。

```
gcc 8.8.c
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示。

```
a[0][0]=10  a[0][1]=11  a[0][2]=12  a[0][3]=13  a[0][4]=14
a[1][0]=15  a[1][1]=16  a[1][2]=17  a[1][3]=18  a[1][4]=19
a[2][0]=20  a[2][1]=21  a[2][2]=22  a[2][3]=23  a[2][4]=24
a[3][0]=25  a[3][1]=26  a[3][2]=27  a[3][3]=28  a[3][4]=29
```

【代码解析】代码第 05 行定义二维数组，代码第 06~13 行用 for 循环的嵌套实现为二维数组元素赋值，代码第 14~22 行用 for 循环的嵌套打印二维数组的元素值。

8.4.2　二维数组与指针

同一维数组一样，二维数组在定义时也返回了一个指向第一个元素的指针。而指针向后移动存储单元，可以访问数组中的元素。下面的代码定义了一个二维数组，然后将二维数组的头指针赋值给一个指针变量。

```
int a[4][5];
int *p;
p=a;
```

根据数组与指针的关系，可以用表 8.4 中的指针变量来访问数组中的所有变量。

表 8.4　二维数组与指针

列 / 行	0	1	2	3	4
0	p	p+1	p+2	p+3	p+4
1	p+5+0	p+5+1	p+5+2	p+5+3	p+5+4
2	p+5+5+0	p+5+5+1	p+5+5+2	p+5+5+3	p+5+5+4
3	p+15+0	p+15+1	p+15+2	p+15+3	p+15+4

从表 8.4 可知，数组中变量的指针地址与首指针存在着下面的关系。

q=p+(行数*总列数)+列数

根据这个关系，可以结合范例 8-9 所示的程序用指针的方法来访问一个数组。

【范例 8-9】使用指针访问二维数组元素，实例代码如下所示。

实例代码 8-9　\源文件\08\8.9.c

```
#include <stdio.h>
int main()
{
   int i=10 , m , n ;                                  /*定义 3 个变量。*/
   int a[4][5];                                        /*定义 1 个 4 行 5 列的二维数组。*/
   int *p;                                             /*定义一个指针。*/
   for(m=0;m<4;m++)                                    /*行的循环。*/
   {
      for(n=0;n<5;n++)                                 /*列的循环。*/
      {
         a[m][n]=i;                                    /*对数组变量进行赋值。*/
         i++;                                          /*i 自加。*/
      }
   }
   p=a;                                                /*把数组的头指针赋值给 p。*/
   for(m=0;m<4;m++)                                    /*行循环。*/
   {
      for(n=0;n<5;n++)                                 /*列循环。*/
      {
        printf("a[%d][%d]=%d  ",m,n,*(p+(5*m)+n));/*用指针来访问数组的变量。*/
      }
      printf("\n");                                    /*每行输出一个换行。*/
   }
   return 0;
}
```

【执行结果】输入下面的命令，编译这个程序。

```
gcc 8.9.c
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序的运行结果如下所示。

```
a[0][0]=10  a[0][1]=11  a[0][2]=12  a[0][3]=13  a[0][4]=14
a[1][0]=15  a[1][1]=16  a[1][2]=17  a[1][3]=18  a[1][4]=19
a[2][0]=20  a[2][1]=21  a[2][2]=22  a[2][3]=23  a[2][4]=24
a[3][0]=25  a[3][1]=26  a[3][2]=27  a[3][3]=28  a[3][4]=29
```

从程序的结果可知，用指针来访问数组，与用下标访问数组的效果是一样的。

【代码解析】代码第 20 行中表达式“*(p+5*m)+n)”的功能是计算数组元素的地址，“p”数组的首地址，“5*m”为一个含有 5 个元素的数组所占用的内存单元数目，则“p+{5*m}”可以计算出第 *m* 行的首地址，再加上 *n* 则表示第 *m* 行的第 *n* 个元素。



8.5 实例

数组可以在程序中实现数据存储，程序可以使用循环的方法对数组进行访问。本节将讲解数组与指针的编程使用实例。

8.5.1 学生成绩统计实例

本节编写一个成绩输入与统计程序。在程序中输入 6 个人的姓名、各科成绩。然后程序输出每个人的各科成绩、总成绩和平均成绩。程序的代码如下所示。

【范例 8-10】学生成绩统计程序，实例代码如下所示。

实例代码 8-10

```
01  #include <stdio.h>
02  int main()
03  {
04     char name[6][10];                          /*定义 6 个人的姓名，6 个字符数组。*/
05     int score[6][5];                           /*定义一个二维数组存放成绩。*/
06     int i;                                     /*定义一个计数器变量。*/
07     for(i=0;i<6;i++)                           /*进行 6 次循环。*/
08     {
09       printf("please input the name:\n");     /*提示输入姓名。*/
10       scanf("%s",name[i]);                     /*输入一个人的姓名。*/
11       printf("score ,math:\n");
12       scanf("%d",&score[i][0]);                /*输入数学成绩。*/
13       printf("score ,English:\n");
14       scanf("%d",&score[i][1]);                /*输入外语成绩。*/
15       printf("score ,Chinese:\n");
16       scanf("%d",&score[i][2]);                /*输入汉语成绩。*/
17       score[i][3] =  score[i][0]+score[i][1]+score[i][2]; /*求出这个人的总分。*/
18       score[i][4]= (int)(score[i][3] /3);      /*求出这个人的平均分。*/
19     }
20      printf("result:\n math  English Chinese  total  average :\n");
21     for(i=0;i<6;i++)                           /*循环输出成绩。*/
22     {
23       printf("%s: %d   %d    %d    %d   %d  \n",name[i],score[i][0],score[i][1],
score[i][2],
```

```
24   score[i][3],score[i][4]);                    /*输出一个人各科的成绩。*/
25      }
26      return 0;
27   }
```

【执行结果】输入下面的命令，编译这个程序。

```
gcc 8.10.c
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序运行时，在文本提示后面输入每个人的姓名和各科分数。程序显示的结果如下所示。

```
math  English Chinese  total  average :
jim: 45   76    87    208    69
tom: 56   76    45    177    59
bill: 56   76    87    219    73
lucy: 45   65    87    197    65
bod: 56   76    46    178    59
lily: 34   65    67    166    55
```

【代码解析】实例代码 8-10 的说明如下：

代码第 05 行定义了一个 6 乘 5 的二维数组，共 6 行（每行表示一个学生的各科成绩），每行 5 个元素（分别表示前三门课的成绩、总成绩和平均成绩）。

8.5.2　冒泡法排序实例

冒泡法排序实例是计算机编程中的一个经典算法，实现多个数值的排序。方法是多次循环进行比较，将大数。每次循环时，找出剩余变量里的最大值，然后减小查询范围。这样经过多次循环以后，就完成了对这个数组的排序。这种排序的算法可用图 8-1 来表示。

冒泡法排序使用了反复循环和比较的算法，执行了下面这些步骤。

① 在第一次循环时，拿第 1 个数与第 2 个数进行比较，如果第 1 个数小于第 2 个数，就用一个中间变量使这两个数交换。这样就使第 1 个和第 2 个数从大到小排列。

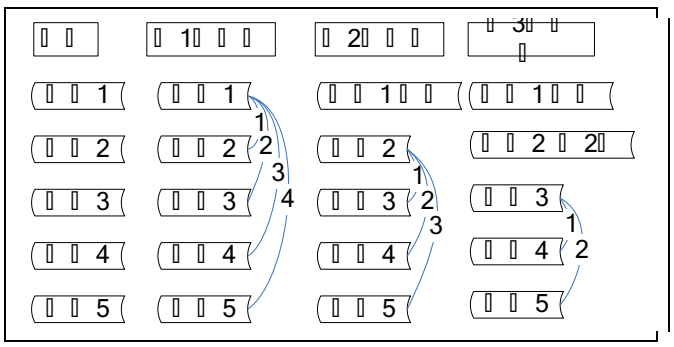


图 8-1　冒泡法排序的算法

② 再用第 1 个数与第 3 个数比较，使这两个数从大到小排列。用同样的方法，用第 1 个数与后面所有的数进行比较。

③ 经过了第 1 轮循环比较以后，第 1 个数一定是所有数里面最大的数。

④ 进行第 2 轮比较。这时让第 2 个数与后面所有的数比较，使用这个数是这些数里面的最大数。

⑤ 用循环的方法，依次用后面的一个数与这个数后面的所有数进行比较，这样就完成了这些数的从大到小排序。

【范例 8-11】用 for 循环语句实现冒泡法排序。下面的程序是使用冒泡法排序的例子。

在这个程序中，需要注意两次循环比较。

实例代码 8-11

```
01  #include <stdio.h>
02  int main()
03  {
04    int a[10];                          /*定义一个整型数组。*/
05    int i,j,temp;                       /*定义循环变量和中间变量。*/
06    for(i=0;i<10;i++)                   /*进行循环输入变量。*/
07    {
08      printf("please enter a number:\n");     /*输出提示。*/
09      scanf("%d",&a[i]);                /*输入变量赋值给数组变量。*/
10    }
11    for(i=0;i<10;i++)                   /*进行 10 次循环。*/
12    {
13      for(j=i+1;j<10;j++)               /*循环比较剩余的变量。*/
14      {
15        if(a[i]<a[j])                   /*如果前面一个数比后面数小，交换两个数的值。*/
16        {
17          temp=a[i];
18          a[i]=a[j];
19          a[j]=temp;
20        }
21      }
22    }
23    for(i=0;i<10;i++)                   /*循环输出排序以后的结果。*/
24    {
25      printf("%d  ",a[i]);
26    }
27    return 0;
28  }
```

【执行结果】编译并运行这一个程序，根据程序的提示输入 10 个变量，程序会对这些变量进行排序，然后输出排序以后的变量。运行结果如下所示：

```
please enter a number:
9
please enter a number:
10
please enter a number:
3
please enter a number:
4
please enter a number:
8
please enter a number:
6
please enter a number:
0
please enter a number:
1
please enter a number:
7
please enter a number:
6
0  1  3  4  6  6  7  8  9  10
```

【代码解析】

- 代码第 11~22 行使用冒泡法进行排序。读者需要仔细体会冒泡法排序的算法实现原理。
- 代码第 23~26 行打印排序后的结果。

8.5.3　统计字符串中的字符

本节将讲解一个统计字符串中的字符的例子。文本中的字符串，是由不同的大写字母、小写字母、标点符号等不同字符构成的。在程序中可以把文本存储在字符串中，然后分别统计出文本中的各种字符个数然后输出。为了识别这些字符，需要学习 ASCII 码。计算机中的所有字符，都是以 ASCII 码的形式保存在计算机中的。每一个 ASCII 码在内存中占据一个字节。ASCII 码与相对应的序号如图 8-2 所示。

Dec	Char	Code	Dec	Char	Dec	Char	Dec	Char
0		NUL	32		64	@	96	`
1		SOH	33	!	65	A	97	a
2		STX	34	"	66	B	98	b
3		ETX	35	#	67	C	99	c
4		EOT	36	$	68	D	100	d
5		ENQ	37	%	69	E	101	e
6		ACK	38	&	70	F	102	f
7		BEL	39	'	71	G	103	g
8		BS	40	(	72	H	104	h
9		HT	41	)	73	I	105	i
10		LF	42	*	74	J	106	j
11		VT	43	+	75	K	107	k
12		FF	44	,	76	L	108	l
13		CR	45	-	77	M	109	m
14		SO	46	.	78	N	110	n
15		SI	47	/	79	O	111	o
16		DLE	48	0	80	P	112	p
17		DC1	49	1	81	Q	113	q
18		DC2	50	2	82	R	114	r
19		DC3	51	3	83	S	115	s
20		DC4	52	4	84	T	116	t
21		NAK	53	5	85	U	117	u
22		SYN	54	6	86	V	118	v
23		ETB	55	7	87	W	119	w
24		CAN	56	8	88	X	120	x
25		EM	57	9	89	Y	121	y
26		SUB	58	:	90	Z	122	z
27		ESC	59	;	91	[	123	{
28		FS	60	<	92	\	124	\|
29		GS	61	=	93	]	125	}
30	▲	RS	62	>	94	^	126	~
31	▼	US	63	?	95	_	127	⌂

图 8-2　ASCII 码表

字符的 ASCII 码值与整型变量是等同的，这就很容易把字符与整型变量联系起来，通过序号的比较可以判断这些字符的种类。从图 8-2 可知，字符与序号的分布可以总结成下面的分段。

- 48～57：数字字符。
- 65～90：大写字母。
- 97～122：小写字母。
- 其他字符：特殊符号或标点符号。

要完成字符的统计，需要在程序中完成下面这些步骤。

① 定义一个字符数组，用来存储字符串。
② 定义相关的计数变量。变量要赋初值为 0。
③ 从键盘输入一个字符串，这一操作可以用 scanf()函数实现。
④ 用循环的方法，访问字符串中的每一个字符。
⑤ 对每一个字符判断字符的类型，然后改变计数的值。
⑥ 用 printf()函数输出统计结果。

根据这些步骤，编写的程序代码如下所示。

【范例 8-12】统计输入的字符，实例代码如下所示。

实例代码 8-12

```
#include <stdio.h>
int main()
{
  char a[100];                                  /*定义一个字符数组保存字符串。*/
  int i,uper,lower,marks,num;                   /*定义计数的变量。*/
  i=0;                                          /*计数变量赋初值为 0。*/
  uper=0;
  lower=0;
  marks=0;
  num=0;
  printf("please input a string:\n") ;          /*提示输入信息。*/
  scanf("%s",a);                                /*输入一个字符串。*/
  while(a[i]!=NULL)                             /*循环访问字符串中的第一个字符。*/
  {
   if(a[i]<=57&&a[i]>=48)                       /*判断是否是数字。*/
   {
     num++;                                     /*是数字则数字计数加 1。*/
   }
   else                                         /*其他的可能。*/
   {
     if(a[i]<=90&&a[i]>=65)                     /*判断是否是大写字母。*/
     {
       uper++;                                  /*是小写字母则大写字母的记数加 1。*/
     }
     else                                       /*其他的情况。*/
     {
       if(a[i]<=122&&a[i]>=97)                  /*判断是否是小写字母。*/
       {
         lower++;                               /*是小写字母则小写字母的记数加 1。*/
       }
       else
       {
         marks++;                               /*其他的情况一定是特殊字符加 1。*/
       }
     }
   }
   i++;                                         /*字符总数加 1，进行下一次循环。*/
  }
  printf("total chars: %d \n",i) ;              /*输出总字符数。*/
  printf("uper chars: %d \n",uper) ;            /*输出大写字符数。*/
  printf("lower chars: %d \n",lower) ;          /*输出小写字符数。*/
  printf("number chars: %d \n",num) ;           /*输出数字字符数。*/
  printf("other chars: %d \n",marks) ;          /*输出其他字符数。*/
  return 0;
}
```

【执行结果】输入下面的命令，编译这个程序。

```
gcc 8.12.c
```

输入下面的命令，运行这个程序。

```
./a.out
```

程序提示输入一个字符串，这时输入下面的字符串，然后按“Enter”键。需要注意的是，字符串中不要有空格。

```
asdfghJKLMNQWERT123789123;',./ASDJjhjad
```

程序显示出的统计结果如下所示。

```
total chars: 39
uper chars: 14
lower chars: 11
number chars: 9
other chars: 5
```

【代码解析】代码第 15 行、第 21 行和第 27 行判断输入的字符类型，它们是利用字符在计算机内保存的是其 ASCII 值的特点来实现的。

8.6　小结

数组与指针的知识是 C 程序中的重要内容，使用它们可以实现很多复杂的功能和算法。在数组与指针的知识中，这两个概念的理解是个难点，需要建立起直观形象的空间概念。在实际开发时，各种对象与算法常常是通过指针与数组建立起来的，需要使用各种复杂的指针和数组。在程序设计练习时，需要注意数组与指针的应用技巧。

8.7　习题

1．编写一个成绩输入与统计程序。在程序中输入 5 个人的姓名、各科成绩。然后程序输出每个人的各科成绩、总成绩和平均成绩。

2．编写小写字母转换的程序。输入一个字符串存放到一个数组中，转换数组中的小写字符为大写字符，其他字符不变，存放到新的字符数组中。原字符数组内容保持不变。

3．定义一个二维字符数组，最多可以存放 10 个字符串，每个字符串最大长度为 128 字节。从键盘输入最多不超过 10 个字符串，然后输出到屏幕。

8.8　面试题

【题目 1】请分别指出以下两个程序的错误。

第一个程序：

```
int a[60][250][1000],i,j,k;
for(k=0;k<=1000;k++)
     for(j=0;j<250;j++)
          for(i=0;i<60;i++)
               a[i][j][k]=0;
```

第二个程序：

```
swap( int* p1,int* p2 )
{
  int *p;
  *p = *p1;
  *p1 = *p2;
  *p2 = *p;
}
```

【分析】第一个程序考查多维数组的赋值时的下标问题；第二个程序考察是否了解野指针。

【题目 2】数组 a[N]存放了 1 至 *N*-1 个数，其中只有一个数字重复存在。请编写一个函数，找出重复存在的数，打印出重复数字的重复次数。函数声明如下：

```
int dump_number(int a[],int N);
```

【分析】此题考查遍历数组及基本的比较算法的掌握程度。

【题目 3】请写出以下程序的输出结果。

```
int main()
{
int a[5]={1,2,3,4,5};
int *ptr=(int *)(&a+1);

printf("%d,%d\n",*(a+1),*(ptr-1));
}
```

【分析】此题考查对指针和数组内在含义的理解。

Broadview®
www.broadview.com.cn
博文视点·IT出版旗舰品牌
技术凝聚实力·专业创新出版

精彩内容，尽在21天学编程系列

本书涵盖主题

- Linux编程平台
- Linux常用命令
- bash使用
- C语言基础
- 字符和字符串处理
- 文件和目录
- 网络编程
- 图形界面编程
- 信号与事件
- 文本编辑器vim、gvim、gedit
- shell编程
- 编译和调试
- 数组与指针
- 结构体
- 文件I/O
- 套接字通信
- 界面布局

21天学编程系列



上架建议：Linux C

ISBN 978-7-121-17151-2

9 787121 171512 >

定价：59.80元（含DVD光盘1张）

策划编辑：胡辛征
责任编辑：李利健
封面设计：李　玲